AD-A046 818   DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC   F/G 9/2
COMRADE DATA MANAGEMENT SYSTEM CONVERSATIONAL INTERFACE USERS M--ETC(U)
JAN 76   W C GORHAM, S E WILLNER, R J MARTIN

UNCLASSIFIED   DTNSRDC-76-0007   NL

1 OF 2
AD A046818

Report 76-0007

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Md. 20084

B.S.

COMRADE DATA MANAGEMENT SYSTEM CONVERSATIONAL INTERFACE USERS MANUAL

by

William C. Gorham, Jr.
Stanley E. Willner
Roger J. Martin
Mark S. Shaw
Michael A. Wallace

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

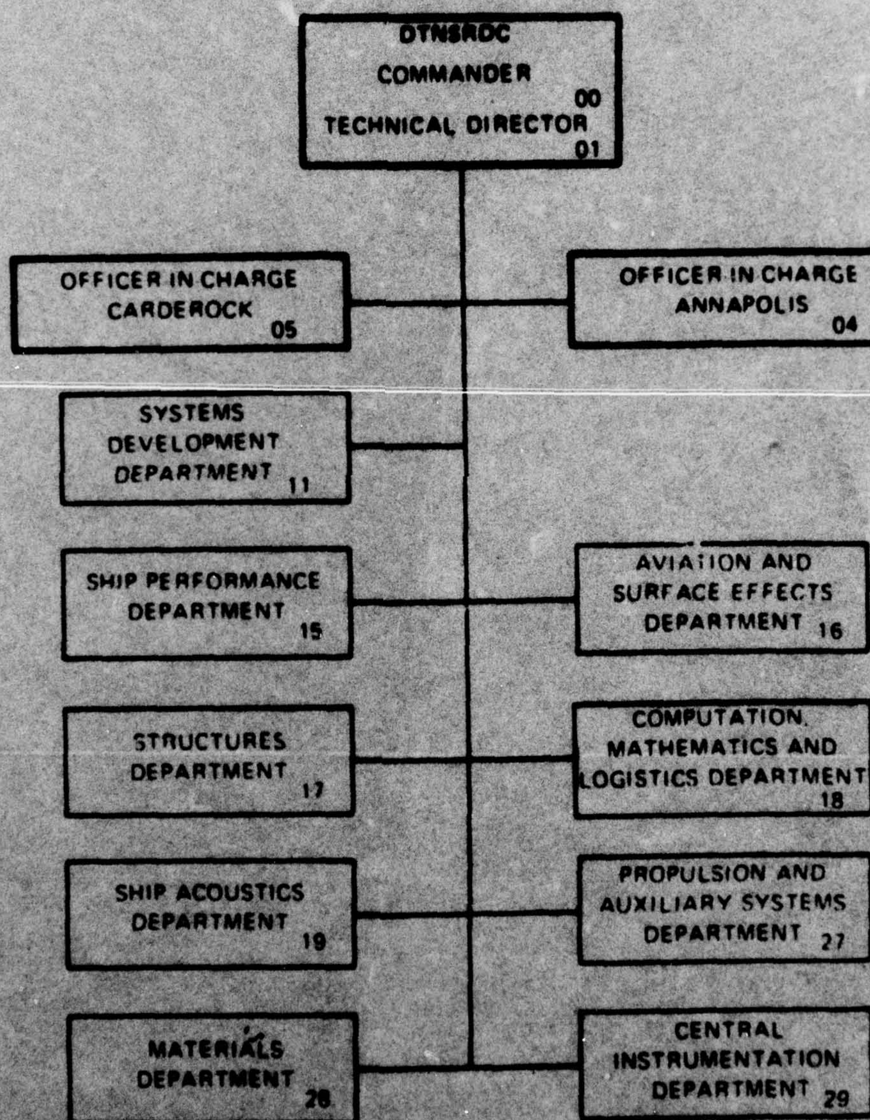
COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

January 1976

Report 76-0007

COMRADE DATA MANAGEMENT SYSTEM
CONVERSATIONAL INTERFACE USERS MANUAL

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Report 76-0007

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): 6 COMRADE Data Management System Conversational Interface Users Manual.

5. TYPE OF REPORT & PERIOD COVERED:

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): 10 William C. Gorham, Jr., Mark Shaw, Stanley E. Willner, Michael A. Wallace, Roger J. Martin (5)

8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: David W. Taylor Naval Ship R&D Center, Bethesda, Maryland 20084

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63509N Project S0331H, Task 14525, Work Unit 1850-030

11. CONTROLLING OFFICE NAME AND ADDRESS: 14 DTNSRDC-76-0007

12. REPORT DATE: 11 January 1976

13. NUMBER OF PAGES: 95

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 9 Research and development rept.,

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

12 95p. 16 S0331H 17 S0331H

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COMRADE	Data definition	Data base administration
Data base management	Data maintenance	Conversational interface
Self-contained system	Report generation	Data dumping
Host Language Interface	Query processing	Data base statistics
Data Loading	Hashing technique	Computer-aided design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The COMRADE Data Management System (CDMS) is a generalized data base management system that has been developed as an independently useful part of the Computer-Aided Design Environment (COMRADE) System. CDMS consists of two distinct components: a set of host language subroutines which enable CDMS functions to be embedded within FORTRAN Extended and COBOL application programs; and a set of conversational programs which enable a user seated at a keyboard/printer or display terminal to define, load, update, and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

387 682

LB

query a CDMS data base, to issue a variety of file management commands, and to access a number of auxiliary capabilities. This report provides an overview of CDMS, and describes the procedural steps involved in the use of the CDMS conversational interface. CDMS operates on the DTNSRDC Control Data 6700 computer.

DEPARTMENT OF THE NAVY

DAVID W. TAYLOR NAVAL SHIP
RESEARCH AND DEVELOPMENT CENTER
BETHESDA, MARYLAND 20084

COMRADE DATA MANAGEMENT SYSTEM
CONVERSATIONAL INTERFACE USERS MANUAL

by

William C. Gorham, Jr.
Stanley E. Willner
Roger J. Martin
Mark S. Shaw
Michael A. Wallace

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

Prepared for
The Naval Ship Engineering Center
Code 6105B

Developed for
CASDAC
The Computer-Aided Ship Design and Construction Project

CASDAC No. 520523/APAC
UM-06

January 1976

Report 76-0007

FOREWORD

The comprehensive Computer-Aided Design Environment (COMRADE) software system has been developed at the David W. Taylor Naval Ship Research and Development Center and sponsored by the Naval Sea Systems Command (NAVSEA, formerly NAVSHIPS) in support of their Computer-Aided Ship Design and Construction (CASDAC) effort to facilitate the design and construction of Naval vehicles. Although conceived in support of ship design, the COMRADE software has been developed for use as a general design tool. Three separate components are included: the COMRADE Executive System; the COMRADE Data Management System; and the COMRADE Design Administration System. All of these components are operable on the DTNSRDC Control Data 6700 computing system (SCOPE 3.4 operating system) and are documented in a set of eight DTNSRDC reports:

- COMRADE - The Computer Aided Design Environment Project, An Introduction
- COMRADE Executive System Users Manual
- COMRADE Data Storage Facility Users Manual
- COMRADE Absolute Subroutine Utility Users Manual
- COMRADE Data Management System Primer
- COMRADE Data Management System Host Language Interface Users Manual
- COMRADE Data Management System Conversational Interface Users Manual
- COMRADE Design Administration System Users Manual

Inquiries concerning any of the components of the COMRADE system should be directed to the Computer Systems Development Group, Computer Sciences Division, DTNSRDC.

It is understood and agreed that the U.S. Government shall not be liable for any loss or damage incurred from the use of these computer programs.

TABLE OF CONTENTS

PRECEDING PAGE BLANK-NOT FILMED

LIST OF FIGURES

I. INTRODUCTION TO THE COMRADE DATA MANAGEMENT SYSTEM

A. SYSTEM CLASSIFICATION AND ORIENTATION

Generalized data base management systems might be classified as either self-contained or host language interface types. A self-contained system typically provides all essential capabilities for data definition, data maintenance, data retrieval, and report formatting. In addition, a self-contained system usually provides a generalized retrieval language that can be used to express fairly complex selection criteria. In contrast, a host language system provides (1) a capability for data definition and (2) an interface, composed of a set of high-level calls, which may be embedded in application programs written in procedure languages such as FORTRAN or COBOL. It is via specialized application programs that the end user of a host language data base performs data maintenance, query processing, and report formatting procedures. Although a host language system always provides an application program interface to the data base, such an interface may also be included in a self-contained system as has been done by the data management component of the Computer-Aided Design Environment (COMRADE) System.

The Computer Aided Design Environment (COMRADE) system developed at the David W. Taylor Naval Ship Research and Development Center (DTNSRDC) in support of the Computer-Aided Ship Design and Construction (CASDAC) Project provides a self-contained type of data base management system which includes

the conversational interface described in the pages that follow. This system, called the COMRADE Data Management System (CDMS) is designed to enable non-computer personnel to define, load, update, and query a data base. Moreover, it supports a host language interface to FORTRAN, COBOL, and COMPASS assembly language programs, so that when specialized needs arise--as frequently happens with CASDAC application software, for example--application programs can be developed to interact as necessary with the data base. The host language interface is therefore an important component of CDMS.

CDMS was developed in the FORTRAN Extended programming language, and is intended primarily for use in scientific applications. It was designed to facilitate management of numeric information used in applications involving the design and construction of ships, aircraft, buildings, dams, and bridges, among others. Although CDMS does provide for the use of textual values, it does not facilitate query and maintenance of textual data as would be required for bibliographic and legal applications.

CDMS is operational under the SCOPE 3.4 operating system[1] on the DTNSRDC Control Data 6700 computing facility (Figure 1). CDMS was developed by, and is maintained and enhanced by, the Computer Systems Development Group, Computer Sciences Division, DTNSRDC.

---

1 References are listed on page 91 in the order of their use.

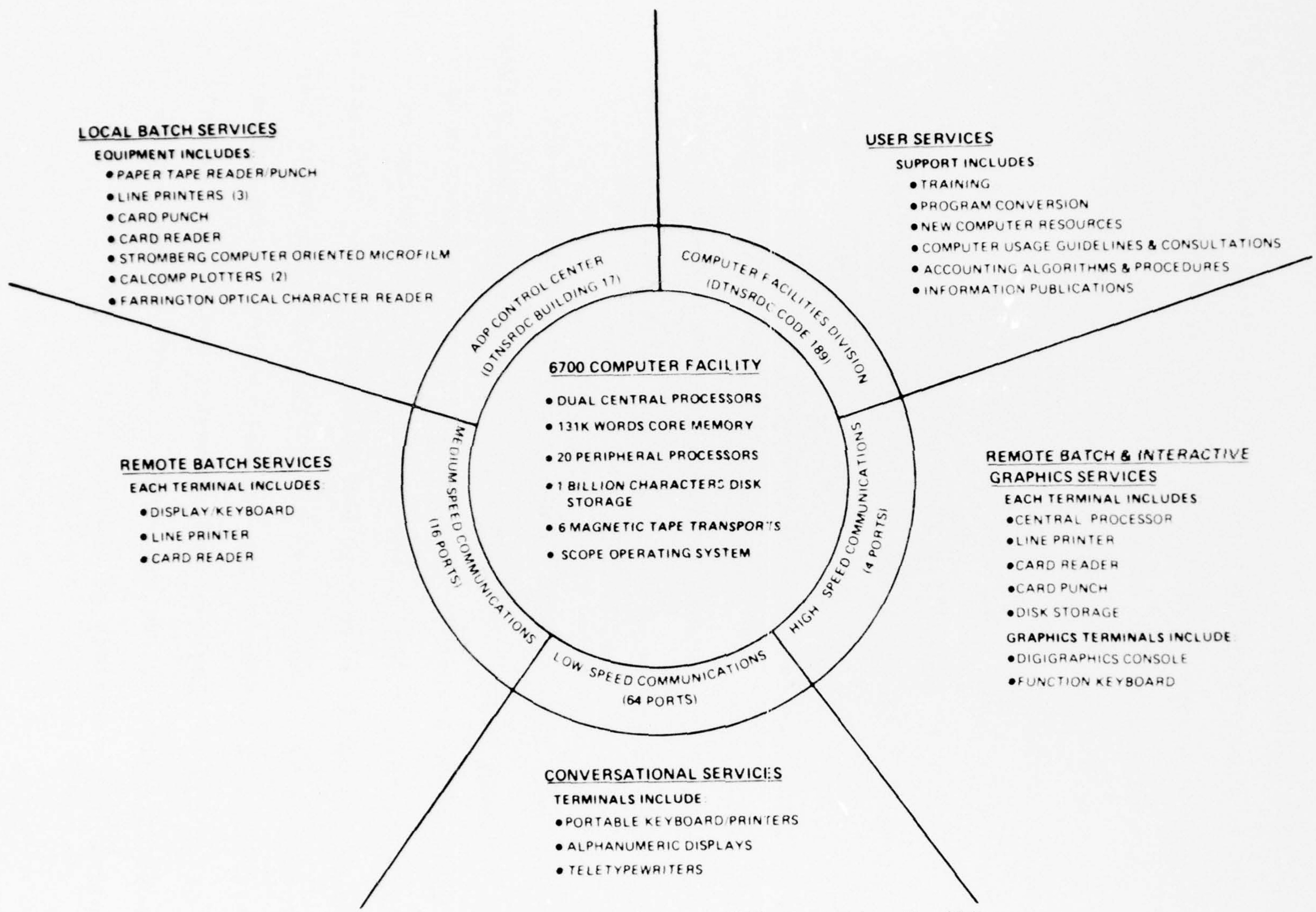


Figure 1 - DTNSRDC Control Data 6700 Computing Facility

B. TRI-LEVEL ARCHITECTURE

CDMS is hierarchically structured into three levels. Users are permitted access to their data at any of the three levels. The flexibility provided by such a design permits the individual user to make his own trade-offs between ease of use of the system and computational efficiency.

The lowest level CDMS component, the COMRADE Data Storage Facility (CDSF),[2] supports storage, retrieval, modification, and deletion of named variable-length logical records or data blocks, and the optional building and processing of inverted data lists for retrieval of information based on data values. The subroutines of CDSF are generally used by the system programmer who values economy of computer time and space above all else. CDSF constitutes the foundation for higher-level CDMS components.

The next higher level CDMS component comprises a set of host language subroutines[3] which serve as the interface between an application program's data requirements and CDSF. It is at this level that logical items of information and groups of information may be retrieved and updated by name. Information may also be retrieved by way of queries involving keyed data attributes. Variable-length logical records may be defined as groupings of single-valued elements, arrays, and repeating groups of elements. Pointer-type elements may be used to link records into a variety of logical data structures.

The third, and highest, level CDMS component is provided to make the system most usable to non-computer-oriented personnel. This component comprises a set of modules suitable for use in batch mode but designed primarily for conversational use. In the conversational mode, a user seated at a keyboard/printer or display terminal may define, load, update, and interrogate his data base. As with the host language interface, the conversational interface allows a user to work with logical items of information. While performing data base retrieval, a user specifies his selection criteria in English-like phrases. All of the CDMS conversational commands are available to any valid user of DTNSRDC's INTERCOM time-sharing system.[4]

The performance of data base definition and processing at the three levels of CDMS are depicted in Figure 2.

C. BASIC CONCEPTS AND TERMINOLOGY

. DATA BLOCK. The basic unit of data storage within a CDMS data base is the data block. Each data block is assigned a unique name of as many as eight characters, the first of which must be a letter, which is used in all retrieval and update transactions involving information in that block; thus for example, in a data base of information about U.S. Presidents, data pertaining to John F. Kennedy might conceivably be collected into a data block given the name KENNEDY; similarly, within a ship design file, information about a particular deck would likely be grouped into a data block given a name such as DK03.

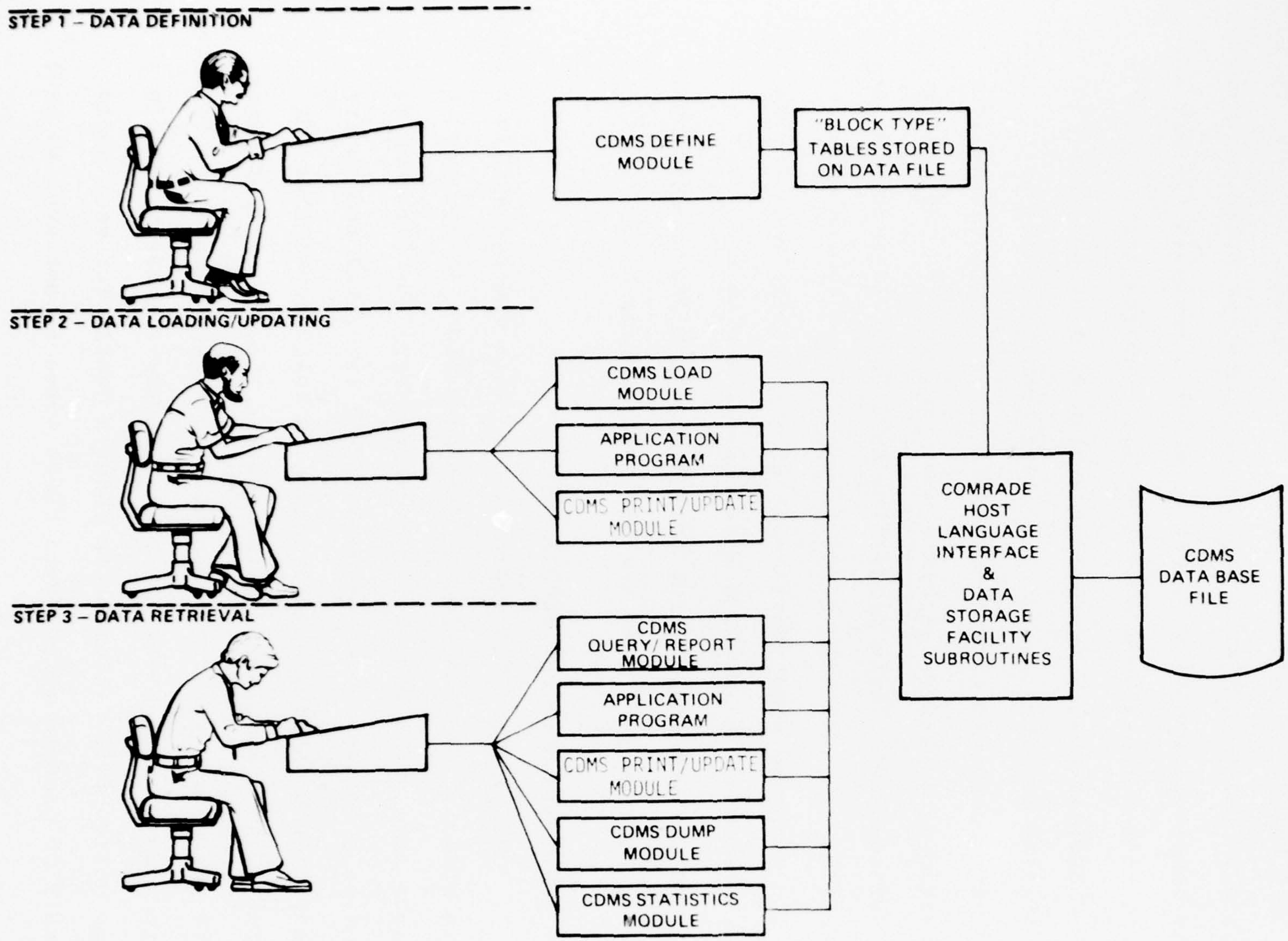


Figure 2 - Data Base Definition and Processing Using CDMS

. BLOCK TYPE. A CDMS data base file comprises one or more sets of data blocks. Each set has a unique record format or definition called a "block type" description. This description gives the logical structure of all data blocks of that particular block type. The description defines subblocks, repeating groups, and data elements and defines the data type (i.e., single or multi-valued, alphanumeric, pointer, real and integer) and inversion status of each element. A U.S. Presidents data base, for example, might have 38 data blocks of the block type PRES (personal information about the Presidents) and 48 data blocks of the block type ELECTION (information pertaining to presidential elections).

To further illustrate the block type concept, assume that the personnel department of a large installation has a data base containing personal data on each of the installation's employees. Personal data for an employee named Tom Jones might be stored in a data block named JONES, and personal data for an employee named John Smith stored in a data block named SMITH. Although this data base may include many data blocks, all blocks storing personal data will be of the same block type perhaps named EMPLOYEE. The block-type description for the EMPLOYEE blocks indicates the kinds of data values contained in that set of data blocks. The description may for instance, indicate that the third data value in the data blocks JONES, SMITH, etc., is the integer value for the data element AGE, and that

the fifth data value in each of these data blocks is the alphanumeric value for the element BIRTHDAY.

. SUBBLOCK. For convenience in organizing and retrieving data elements, a data block can be subdivided into uniquely named groups of data elements known as subblocks. Subblocks may be used to group logically related elements and to provide access to subsets of data values within a data block. Each data block may contain as many as eight subblocks. A subblock name may be as many as eight alphanumeric characters long, the first always a letter.

. DATA ELEMENT. In CDMS, the data element is the smallest unit of data that can be defined and referred to by name (the name to be as many as eight characters long, with the first a letter). There are three classes of data elements -- numeric, alphanumeric and pointer:

(a) A numeric data element may contain either real or integer values, and may be either a single-valued quantity (represented by a single 60-bit CDC 6000 computer word) or a variable-length array of such values. If an array of values is given, the data element name will refer to all of the values in the array. The CDMS host language subroutines do, however, provide a means of referring to a particular array element if its position within the array is provided.

(b) An alphanumeric data element may be either a single value (made up of as many as ten CDC 6000 display code characters), an array of such values, or a variable-length text string.

(c) Pointer data elements are used to construct a data structure (tree, ring, network) that links all related data blocks. A pointer element in one data block may "point" to a different data block on the same data base file, or it may "point" to a data block stored on a different data base file entirely. A pointer data element may be defined either as a single value or an array of values.

. REPEATING GROUP. Within a data block, one to 50 consecutively-defined data elements of the same or different type may be grouped together and defined as a repeating group. A repeating group is identified by a name (as many as eight characters, beginning with a letter) which may be used when referring to the entire grouping of data elements. Each element in the group may, of course, be referenced by its own individual name.

A data block may contain a variable number of repetitions or occurrences of repeating group values. Although there may be at most 50 data elements in a repeating group, the number of occurrences of values for these elements is virtually limitless.

To illustrate the repeating group concept, assume that employee Tom Jones has three children and employee John Smith

has but one child. If in a block type named EMPLOYEE there exists a repeating group named CHILDREN, and if CHILDREN includes the data elements NAME and AGE, then in the data block named JONES there would be three occurrences of CHILDREN (i.e., NAME and AGE) while in the data block named SMITH there would be only one occurrence.

. INVERTED DATA ELEMENT. Quick resolution of conditional retrieval requests or "queries" issued from either conversational terminals or application programs is an important aspect of data base processing. CDMS uses a set of "inverted lists" to minimize query response times. An inverted list is a directory which (1) contains the data values for a particular data element which was assigned inversion status when it was defined, and (2) contains the names of all data blocks having values for the inverted elements. The data-value/block-name pairs are sorted by value.

To illustrate, assume a CDMS data base of personnel data. Assume also that the data base contains data blocks of the names JONES, SMITH, JOHNSON, BROWN, and WILSON, which contain personal data on Tom Jones, John Smith, Jim Johnson, Ed Brown, and Bob Wilson, respectively. If, in the block type EMPLOYEE there is a data element AGE which has been assigned inversion status, values for the element AGE will be stored in an inverted list which might look as follows:

AGE

	25
SMITH	
	28
JOHNSON	
	28
BROWN	
	36
JONES	
	38
WILSON	

If the user requested a printout of the values for the data element WEIGHT (another data element within EMPLOYEE) for all persons at least 30 years of age, the inverted list for AGE would be examined and only those data blocks satisfying the condition - namely, JONES and WILSON - selected for further processing to obtain WEIGHT information.

. LOCALLY DEFINED DATA ELEMENTS. An individual data block may include data elements not logically related to other blocks of the same block type. These locally defined data elements are managed in exactly the same way as the other data elements defined within a block-type description except that they are not considered for membership within repeating groups.

. CROSS FILE REFERENCE TABLE. In addition to containing user-defined data blocks, block types and inverted lists, a CDMS data base file contains a cross-file reference table which lists the permanent file names of all data base files "pointed to" by pointer data elements contained within this particular file. The permanent files listed in the table

may be one of two types: a SCOPE permanent file, indicated by a four-character ID and a file name of as many as 40 characters, or a CPFMS (COMRADE Permanent File Management System) file, indicated by level-three and level-four names. (See Appendix A for a detailed description of the CPFMS file management capability.)

D. SYSTEM FEATURES AND CAPABILITIES

To facilitate the comparison of CDMS features with those of other generalized data base management systems, the following analysis has been patterned after that of the CODASYL Technical Report of May 1971[5] and of the National Bureau of Standards Technical Note 887 of November 1975 (a comparison of six generalized data base management systems).[6] Features are described under eight major headings: computer environment, data structure, data definition, data maintenance, query specification, output and report generation, security features, and external linkages. Each heading is further subdivided as necessary to provide greater detail.

1. Computer Environment

CDMS is operational on the DTNSRDC Control Data 6700 computing facility which is supported by the SCOPE 3.4 operating system. The core requirement for the conversational interface approximates 25K (decimal) words. The core requirement, excluding buffer space, for the host language interface and the COMRADE Data Storage Facility (when used with the COMRADE

Absolute Subroutine Utility)[7] approximates 8K (decimal) words. The source language for most of CDMS is FORTRAN Extended; only a very small number of COMPASS assembly language statements have been included.

The communication link between the terminal user and CDMS is supported by the INTERCOM time-sharing system. CDMS is not coded re-entrantly. Its components may be used in either the batch or the conversational mode.

2. Data Structures

- Data Types

CDMS provides a set of valid data types which can be defined for single-valued and array elements. These valid types are numeric, including real and integer values; alphanumeric, including a single value of as many as ten characters, an array of such values, and a variable-length text string; and pointer. A character may be any letter, digit, or special symbol supported by the hardware (see Appendix B).

- Logical Data Structure

The logical data structure is defined as the composition of data elements in the data blocks, and the interrelationships between data blocks and their files as determined by the user without regard for main memory or secondary storage. CDMS data blocks may contain single-valued and variable-length array elements. Variable-length repeating groups are also permitted. Data may be interrelated across files and among data blocks through the use of "pointers",

thereby providing full network structuring. Moreover, inverted list structures are supported for single-valued elements.

- Physical Storage Structure

Data records within CDMS are randomly organized, and are located by transforming the data block name into the disk address. This type of transformation, or hashing technique, makes it possible to access a block during its residence on a CDMS file by its symbolic name rather than through a numerical index.

3. Data Definition

CDMS provides the user with a block-type or data definition language. This language serves as the mechanism for defining data elements, repeating groups, subblocks, and keyed or inverted elements. The language is card oriented - one card image for each item to be defined. Within a card image, fields may be specified free-form with commas separating the fields.

4. Data Maintenance

- Data Base Loading and Input Analysis

After the user has described his block-types to CDMS, the system uses these descriptions to construct internal block-type tables, one for each description. At data base load time, a block-type table is identified by name, after which one or more data blocks of this particular block type may be created and, optionally, loaded with values. After a data block has been created, a value that is to be loaded into that block must

first be read from a sequentially organized card image file. If the value is numeric, it will be converted into the proper internal format pursuant to the data element definition stored within the block-type table. After conversion, the value is loaded into the data block and referred to in future retrieval and update requests simply by its data element name. A data definition links the card image file of values to a particular set of block types.

- Data Base Restructuring

The restructuring of a data base generally involves the modification of the block type, such as by adding or deleting a data element or repeating group definition. Within CDMS, data base restructuring is permitted so long as no data blocks have been created for a given block type. Restructuring following creation of data blocks involves dumping the existing contents of those blocks, redefining the block type, and re-loading the dumped values. A utility program is available under CDMS for dumping the contents of a data base.

- Data Updates

Updating is the process of adding, modifying or deleting the contents of some part of a data base. Updating differs from data loading in that the load process is typically a batch procedure involving a sizable quantity of values and the creation of new data blocks, whereas the update process is typically a conversational procedure involving transactions

on existing data blocks and the use of fairly small sets of data values. In CDMS, there is a language for specifying update transactions. The language differs from the query language in that it does not involve English-like phrases and conditional operations. The update language does, however, provide for both the updating of and the printing of data elements. The language is card oriented - one card image for each update or print operation. Within a card image, fields may be specified free-form separated by slashes.

- Other Maintenance Capabilities

Lockout during the updating or loading of a CDMS data base is maintained at the file level. As a result, only one user at a time may modify a CDMS data base file. To minimize possible conflicts resulting from concurrent update or load operations, the scheduling of data base modifications may have to be controlled administratively. During a modification procedure, users may retrieve and print data base values.

5. Query Specification

- Overall User-System Interaction and Search Specification

Query processing involves the selective retrieval of data from a data base. Within CDMS, information is entered initially to identify the particular portion of the data base to be interrogated. That information is followed by a set of statements that specify selection criteria and certain

actions to be taken, such as print in columnar or row format, for example. The query language is based upon use of English-like phrases which may contain Boolean combinations (i.e., AND or OR) of relational expressions. A relational expression in most instances is a triple composed of a data element name, a relational operator (i.e., GT, GE, LT, LE, NE or EQ), and one data value; in the case of the operator "between," the expression is composed of a name, the operator BET, and two values.

- Multi-File Searching

The CDMS query capability enables as many as five data base files to be searched initially. Cross-file and intrafile pointer elements may be traversed. With the current "hit" file (a "hit" file is a list of data blocks which have satisfied a specific condition) as a starting point, as many as eight different pointers (specified on a single statement) may be processed by the system. Typically, processing of cross-file pointers will continue without user intervention.

- Additional Query Features

CDMS allows hit files to be named and saved for use in subsequent query statements. Searches based on values of non-inverted data elements may be conducted on the data blocks named in a hit file. A hit file name table, and the number of "hits" on a hit file, will be printed upon request.

A pseudo batch mode is available to the user during conversational query processing. In this mode, the user may submit a set of query statements which will then be checked for syntax

errors; if there are no errors, the statements will be executed without further user interventions.

6. Output and Report Generation

- Language Type and Media Selection Flexibility

An output and report generation capability is provided as a subset of the CDMS Query capability. English-like phrasing is used to specify one of three report formats -- columnar, row, or simple list. Typically, a report is printed on the user's keyboard/printer or display terminal. Lengthy reports can be routed to off-line medium - and high speed printers.

- Arithmetic and Sorting Capabilities

Computations on retrieved numeric vaues are supported within CDMS by a set of built-in functions which include sum, difference, quotient, product, and columnar totals, and averages. Sorting, also supported by a built-in function, may be specified in columnar-report requests and may be performed on as many as eight fields in accordance with the order indicated (i.e., LOW A, HIGH B, LOW C, etc.)

7. Security Features

- File Protection and Access Control

A CDMS data base file must be given a name (of as many as 40 characters), and may be password protected in accordance with SCOPE 3.4 operating system conventions. The file is protected from unauthorized access by way of a set of privacy controls

specified at the time the file is cataloged into the SCOPE system. Since access privileges may vary from user to user, a variety of controls are provided including read only, read and modify, increase mass storage allocation, and delete a file from the SCOPE file system.

An additional level of CDMS data base file protection may be imposed by way of the COMRADE Permanent File Management System (CPFMS). A detailed description of the CPFMS capability is provided in Appendix A.

- File Backup, Recovery, and Restart

At DTNSRDC, safeguards exist to prevent an on-line CDMS file from being accidentally destroyed during normal system operation and normal loading of the operating system. For backup purposes and to guard against data loss as a result of some of normal system event, on-line files are generally dumped to magnetic tape, on a semi-weekly basis. A variety of utility programs exist at DTNSRDC for use in more frequent dumping of data base files. Among these is the CDMS dump module, discussed in subsequent sections of this report.

8. External Linkages

As mentioned previously, CDMS supports a host language interface to COBOL,[8] FORTRAN Extended,[9] and COMPASS[10] application programs. These programs request services of the host language subroutines through control statements (such as the CALL statement of FORTRAN Extended, for example).

E. DATA BASE ADMINISTRATION

In an environment such as computer-aided ship design, in which the data base includes data to be shared by many users and programs, it becomes necessary for certain functions to be centrally coordinated and controlled. In such an environment the data base must inevitably reflect a compromise between the needs of the various competing users and their programs. Conflicts resulting from such diverse needs must be satisfactorily mediated and resolved. Generally this role of arbitrator is performed by a data base administrator (DBA) who possesses data processing experience, communications skills, and a working familiarity with user operations. The DBA is responsible for functions spanning three broad categories of data base design and management: organization, monitoring, and reorganization.

- Organization

During the organization of a data base, the DBA assesses the needs of the competing users and programs. Following a determination of all the data requirements, the DBA will (a) use the block-type definition language to define the logical structure of the data base including those subblocks, repeating groups, and data elements that serve to model the problem with which the data base will be concerned; (b) coordinate data base loading and updating activities; (c) assign data base file names and access controls, taking into account the access privileges of all data base users; and (d) coordinate the activities of

specialists working to organize the data base.

- Monitoring. During the life cycle of a data base, it is the DBA's responsibility to develop strategies for monitoring the use of the data base, breach of privacy, frequency of backup, and the need for reorganization.

- Reorganization. As a result of information obtained from the monitoring procedure, or because of the changing data requirements of the users and their programs, the DBA may find that the data base must be reorganized. The DBA may (a) change the block-type descriptions; (b) reassign access controls; (c) alter the frequency of backup operations; (d) change the data base contents to reflect changes in the block types and the logical data structure; and (e) remove from the data base any allocated storage space not currently being used productively.

These activities indicate the kinds of functions for which the DBA and his staff are responsible. Since many of the user languages supported by CDMS are oriented towards non-computer personnel, they are most appropriate for use by the DBA and any persons working under his direction in that they allow operations to be readily formulated and concisely stated.

F. POSSIBLE COURSE OF FURTHER SYSTEM DEVELOPMENT

As the computer-aided ship design systems of CASDAC progress from the planning stage to the developmental stage,

augmentation of CDMS will likely be required. This effort would be undertaken along two fronts: enhancements of CDMS features and capabilities on the DTNSRDC computing facility, which is to eventually become an operating node of the Navy laboratory computing network; and development of a subset of CDMS features and capabilities on a set of minicomputers which are to be located at the Naval Ship Engineering Center and linked to the DTNSRDC computer.

II. OVERVIEW OF THE CDMS CONVERSATIONAL INTERFACE

A. ROLE OF THE INTERFACE

The CDMS conversational interface serves to link the user--perhaps a data base administrator (DBA), an engineer, or a non-computer-oriented manager seated at a keyboard/ printer or display terminal--with his data base and a comprehensive set of data management tools that are operable on the DTNSRDC computer. Interactive techniques enable the user to perform his data base analysis, and to evaluate--and, if necessary, rectify--the outcome immediately, without undergoing the delay and inevitable interruption to his train of thought that is associated with operating in the batch mode. The conversational interface enables the user to define, load, update, and query a data base file. Moreover, it enables him to generate certain types of reports; obtain statistical information about data blocks, block types, and inverted lists; issue a variety of file management commands; and access a number of auxiliary capabilities such as the INTERCOM EDITOR program.

B. FILE MANAGEMENT

The CDMS conversational interface comprises a set of six program modules. To facilitate file management among these modules--each module is activated by way of a module execution command--CDMS supports a file management command language. Employing this language a user can manipulate files stored

within the DTNSRDC SCOPE 3.4 file system. Specifically, he can (1) attach SCOPE or CPFMS (See Appendix A) files, (2) catalog files, (3) purge files, (4) unload files, (5) activate data base files for subsequent processing by conversational modules, (6) copy data base files, and (7) list the names of all local files currently available for use.

C. DATA BASE DEFINITION

A user defines, modifies, deletes and prints block-type descriptions with the CDMS DEFINE program module. The user of the DEFINE module need not have extensive programming experience. However, he should have a thorough knowledge of the data which will be included in the data base, for he must provide very specific information about each of the subblocks, repeating groups, and data elements for each block type.

During block-type definition, the user must provide a name by which the block type will be referenced in subsequent operations.

D. DATA BASE LOADING AND DUMPING

After the user has defined and named his block type, CDMS uses the definition to construct an internal block-type table. When the user decides to create and load data blocks for the newly defined block type, he calls upon the CDMS LOAD module. After a data block has been created by LOAD, values to be loaded into the block are read from a

sequentially organized card image file. After a value has been read, it is converted, if necessary, and inserted into the data block. If an inverted data element is being loaded, that element's inverted list is appropriately modified. A data definition serves to link the card image file containing only data values with those elements defined in the block type.

The inverse of the LOAD module is the DUMP module. The DUMP module maps a data definition (similar to the one used during loading) against an existing, loaded data base file and produces a sequentially organized card image file of values useful for the restructuring of the data base.

E. DATA BASE UPDATING

With the passage of time, data element values often become obsolete. Moreover, values not known at LOAD time later become available. Accordingly, the UPDATE module provides capabilities for adding, modifying, and deleting individual data element values, and for creating and deleting data blocks. In addition, the UPDATE module provides capabilities for displaying the values of specified portions of the data base (such as entire data blocks or individual subblocks, repeating groups, and data elements).

F. QUERY PROCESSING

Query processing is the selective retrieval of data from a data base. The CDMS QUERY module searches inverted lists to determine which data blocks satisfy the user-imposed conditions. Using a resultant set of data blocks (i.e., a "hit" file) as a basis for further processing, the user can impose additional search criteria, search non-inverted elements, print relevant information in his choice of three report formats, or trace a string of pointers to its terminal point (another set of data blocks).

G. DATA BASE STATISTICS

A CDMS user may occasionally find it helpful to have a profile of the contents of his data base. The STATS module displays statistical information about the data blocks, block types, and inverted lists contained in the data base.

H. AUXILIARY FACILITIES

A number of auxiliary features, activated by a command language, are provided to assist the CDMS user with his conversational operations:

- The INTERCOM EDITOR program
- A facility that causes a given report to be printed at the user's terminal and/or written to a local file.
- A capability which causes a local report file to be routed to off-line medium- and high-speed printers.
- A capability which tutors the CDMS conversational user.

III. DETAILED DESCRIPTION OF THE CONVERSATIONAL INTERFACE

A. ENTERING AND EXITING CDMS

Any valid user of the DTNSRDC INTERCOM time-sharing system may gain access to the CDMS conversational interface simply by entering the command *COMRADE*, *CDMS*. An INTERCOM user already executing within the Computer-Aided Design Environment (COMRADE)[11]--perhaps he has been using some of the COMRADE Executive System's[12] command procedures--need only enter the acronym *CDMS*.

After CDMS has been initiated, the user may issue certain "global" file management, auxiliary, and module execution commands. These commands, termed "global" because they do not relate exclusively to a particular CDMS module, are indicated in the following list:

File Management	Auxiliary	Module Execution
ATTACH	*EDITOR*	*DEFINE*
CATALOG	*OUTPUT*	*LOAD*
PURGE	*SEND*	*QUERY*
UNLOAD	*HELP*	*UPDATE*
COPYDB	*EXIT*	*STATS*
DATABASE		*DUMP*
FILES		

A global command (such as *ATTACH*, for example) might be issued after a user enters the DEFINE module, or after he enters the QUERY module, or perhaps even before he enters any of the CDMS modules. Upon completion of CDMS conversational operations the user need only enter the global command *EXIT*. If the user who has *EXIT*ed from CDMS wishes to leave COMRADE but to remain "logged in" to INTERCOM, he should type the command *SIGNOFF*. The user wishing to terminate his INTERCOM session should type *LOGOUT*.

B. USE OF GLOBAL COMMANDS

1. File Management Commands

ATTACH, lfn, pfn, parameter list.

Any SCOPE permanent file or CPFMS file (see Appendix A) can be rendered accessible to the various CDMS program modules by issuing the *ATTACH* command. The parameter *lfn* is the local file name. The parameter *pfn* is the permanent file name if the file is a SCOPE file; for a CPFMS file, *pfn* must consist of the level-3 name followed by a comma and the level-4 name. The parameter *parameter list* must include the ID and any passwords or permissions needed to grant access to the SCOPE file; for a CPFMS file, the *parameter list* may consist of either of the optional parameters *PW* (denoting pseudo-password) and *MR=1* (as described on page 5-12 of the SCOPE Reference Manual[1]).

Examples:

```
ATTACH, CDMSDB, PRESIDENTSDB, ID=COMR.
ATTACH, SDF, DDG3, DDG3, MR=1.
```

CATALOG, lfn, pfn, parameter list.

Any SCOPE or CPFMS file created when CDMS modules are being used may be cataloged in the computer system. The parameters *lfn* and *pfn* are as described in the *ATTACH* command (see earlier paragraph). The parameter *parameter list* must include the *ID*, the *AC* (account number) and any passwords or permissions needed to control access to the SCOPE file; for a CPFMS file the *parameter list* may consist of either of the optional parameters *PW* (pseudo-password) or *FL* (file access lock; consult the discussion presented in Appendix A).

PURGE, lfn, pfn, parameter list.

A SCOPE or CPFMS file can be removed as a cataloged file through use of the *PURGE* command. The parameters *lfn*, *pfn*, and *parameter list* are as described for the *ATTACH* command. Note that a CPFMS file can be purged only if it has not been attached, and that unlike the purge of a SCOPE file, a CPFMS file will be unloaded after it has been purged.

If a SCOPE file has been attached with the correct permissions, the parameters *pfn* and *parameter list* may be omitted from the *PURGE* command. For further discussion see pages 5-14 and 5-15 of the SCOPE Reference Manual[1].

UNLOAD lfnlist

The *UNLOAD* command allows a user to unload one or more SCOPE or CPFMS files. The parameter *lfnlist* may consist of one or more local file names separated by commas.

Examples:

```
UNLOAD, CDMSDB, PRESDB, SDF.
            or
UNLOAD, CDMSDB.
UNLOAD, PRESDB.
UNLOAD, SDF.
```

DATABASE lfnlist

The *DATABASE* command allows the user to declare a particular file as the "active" data base. The data base to be specified may be either a permanent file which has been attached, a newly created data base, or a data base which the user is about to create. All CDMS program modules will operate on the "active" data base until another *DATABASE* command is issued. For all modules except the QUERY module, the parameter *lfnlist* must consist of only one local file name; when used with the QUERY module, the parameter *lfnlist* may consist of up to five local file names.

COPYDB lfn

The *COPYDB* command allows the user to copy and compress the "active" data base file (i.e., the file referred to in the most recent *DATABASE* command). The parameter *lfn* is the name of the file which is the result of the copy and compression process.

FILES

The *FILES* command allows the user to obtain a print out of the names of all the local files which are available for processing; permanent files will be identified with an asterisk (*).

2. Auxiliary Commands

EDITOR

During the course of execution, certain of the CDMS program modules (DEFINE, LOAD, DUMP) require that "input files" be submitted by the user. The *EDITOR* command allows the user to enter the INTERCOM EDITOR program to create the necessary files and/or to modify files created previously. (See the CDC INTERCOM Reference Manual 6000, Version 4, Chapter II-2.[4])

Plans are underway to enable the user to enter EDITOR directly from any module of CDMS by simply typing in the command *EDITOR*. When he leaves the EDITOR program, he will then be returned to the module from which he entered. However, for the present, the following procedure must be followed. The user types in the command *EDITOR* and receives a message telling him that he is about to leave CDMS. Next, he is put in the INTERCOM "command" mode and he must then again type in the command *EDITOR*. He may then perform any necessary processing within EDITOR, leaving in the normal manner (via the EDITOR command *BYE* or *B*). Once again in the "command" mode, the user must type *CDMS* to return to the program module he was executing when he first typed *EDITOR*.

OUTPUT, *site*

The *OUTPUT* command allows the user to specify whether his output is to be printed at his terminal, written off-line onto a "Report File", or written to both destinations. The parameter *site* must be one of the following: *HERE* (denoting

the user's terminal); *FILE* (denoting the "Report File"); or *BOTH* (denoting both destinations). If no *OUTPUT* command is submitted during a CDMS session, all output will be printed at the user's terminal.

SEND, site

The *SEND* command allows the user to have his "Report File" printed. The "Report File" might contain output from several of the CDMS program modules. The parameter *site* must be one of the following: *6700* (denoting building 17, DTNSRDC); *1700* (denoting building 192, DTNSRDC); *SEC* (indicating the 1700 computer, NAVSEC); or *HERE* (denoting the user's terminal).

HELP

The *HELP* command allows the user in distress to receive instruction on use of the various CDMS commands.

EXIT

The *EXIT* command terminates execution of CDMS conversational operations. Before the user terminates CDMS, he is given the opportunity to *CATALOG* any file created during the session and to *SEND* the "Report File" to be printed. Note that the *EXIT* command is not to be used to leave one CDMS program module and enter another. To enter a different program module the user need merely type in the name of that module.

3. Module Execution Commands

To enter a CDMS program module initially, or to switch from one module to another, the user need merely type in the name of the module he desires. Module names are as follows:

DEFINE

LOAD

UPDATE

QUERY

STATS

DUMP

Detailed descriptions of each of these program modules are provided in the next section.

C. USE OF PROGRAM MODULES

1. The DEFINE Module

The DEFINE module allows the user to print, delete, create and modify block-type definitions on the "active" data base (i.e., the data base referred to in the most recent *DATABASE* command). This data base may be either an existing CDMS data base or one that the user is currently initiating.

Commands of the DEFINE Module

The four commands, *PRINT*, *DELETE*, *CREATE*, and *MODIFY*, supported by the DEFINE module are as follows:

PRINT btname

The *PRINT* command causes the block-type definition named *btname* to be printed. The command *PRINT ALL* causes all block types on the data base to be printed.

DELETE btname

The *DELETE* command causes the block-type definition named *btname* to be deleted. The command *DELETE ALL* causes all block types on the data base to be deleted.

CREATE btname/lfn

The *CREATE* command causes the block-type definition named *btname* to be created and stored on the data base. The parameter *lfn* is the name of the local file that contains the set of block-type definition statements.

MODIFY btname/lfn

The *MODIFY* command causes the block-type definition named *btname* to be replaced by the block-type definition stored on the file *lfn*.

Block-Type Definition Statements

A block-type definition, required as input to the *CREATE* and *MODIFY* commands, may consist of the following three types of statements. Note that the *SB* statement for each subblock to be defined must be followed by the entire set of *EL* statements for that subblock, which in turn must be followed by the entire set of *RG* statements for that subblock.

SB-subname

The *SB* statement denotes the beginning of a subblock definition; the parameter *subname* must be a valid subblock name.

EL-elname, typespec

The *EL* statement is used to define a data element. The parameter *elname* must be a valid data element name; the

parameter *typespec* must be one of the following codes that indicate the data type of an element:

A	Alphanumeric	*P,V*	Inverted pointer
I	Integer	*AA*	Alphanumeric array
R	Real	*IA*	Integer array
P	Pointer	*RA*	Real array
A,V	Inverted alphanumeric	*PA*	Pointer array
I,V	Inverted integer	*T*	Text
R,V	Inverted real		

RG-rgname,elname1-elname2

The *RG* statement is used to define a repeating group. The parameter *rgname* must be a valid repeating group name. The parameters *elname1* and *elname2* identify the data elements that constitute the bounds of the repeating group's membership. If the membership consists of only one element, the form of the statement must be *RG-rgname,elname* where *elname* is the data element name.

Block-Type Definition, an Example

A user wishing to define a set of blocks which are to contain personal information about each of the U.S. presidents might specify the following block type:

```
SB-PERSONAL
EL-SURNAME,A,V
EL-FIRSTNAM,A
EL-INITIAL,A
EL-MONTHB,A
EL-DAYB,I
EL-YEARB,I,V
EL-STATEB,A,V
EL-STATEPTR,P
EL-HEIGHT,A
EL-PARTY,A,V
EL-COLLEGE,A
EL-ANCESTRY,A,V
EL-RELIGION,A,V
EL-OCCUP,AA
EL-MONTHD,A
EL-DAYD,I
EL-YEARD,I
EL-CAUSE,A
RG-NAME,SURNAME-INITIAL
RG-BIRTH,MONTHB-STATEB
RG-DEATH,MONTHD-CAUSE
SB-FAMILY
EL-FATHER,A
EL-MOTHER,A
EL-WIFE,A
EL-MONTHM,A
EL-DAYM,I
EL-YEARM,I
EL-CHILDREN,I
RG-MARRIAGE,WIFE-CHILDREN
SB-HISTORY
EL-ELECTION,PA
EL-ADMIN,PA
EL-CONGRESS,PA
```

2. The LOAD Module

The LOAD module allows the user to create data blocks and load them with values taken from a card image data file (DF) according to the format indicated by a data definition (DD). The LOAD module provides the following features:

- The user can save DD's. A saved DD can be used at a later time to load another set of data blocks and/or dump a set of blocks (see the description of the DUMP module for a discussion of the data base dump process). A saved DD can be printed on request.
- The names of the data blocks to be created can be specified in the input data (i.e., in the DF), in the DD, or on a user-supplied file.
- Since DD's are used in conjunction with block-type definitions, they contain no information about data type, inversion status or subblock names.
- Repeating group sizes (i.e., number of occurrences) and array sizes (number of words) can vary from block to block as the user requires.

Commands of the LOAD Module

Four commands, *USE*, *SAVE*, *UNSAVE* and *PRINT*, are supported by the LOAD module for use in creating and maintaining DD's and for identifying files that are to be used during the data base load process.

USE ddname,dfname[*,dbname*]

The *USE* command initiates the data base load and identifies those files which are to be used as inputs to the process. The parameter *ddname* is either the name of a DD already stored on the data base, or the name of a local file containing a DD.

Other parameters include *dfname*, the name of the local DF file, and *dbname* (optional), the name of the local file that contains the names of those blocks which will be created and loaded.

SAVE ddname

The *SAVE* command causes a DD, existing on a local file identified in the most recent *USE* command, to be stored on the data base so it may be invoked in a subsequent data base load (i.e., subsequent *USE* command). The parameter *ddname* must be a 1- to 8-character alphanumeric string beginning with a letter.

UNSAVE ddname

The *UNSAVE* command causes a previously saved DD to be removed from the data base.

PRINT ddname

The *PRINT* command causes a previously saved DD to be printed.

Data Definition Statements

A data definition (DD) may consist of nine types of statements; these are discussed below.

TYPE btname

The *TYPE* statement indicates that data blocks to be loaded according to the current DD are of block type *btname*. The *TYPE* statement must precede all other statements of a DD.

RECORD length

The *RECORD* statement, in which the parameter *length* is a positive integer less than or equal to 80, indicates the length of the data in each of the DF card images. Thus the statement *RECORD 50* would indicate that data to be loaded are contained in Columns 1-50 of the DF card images; Columns 51-80 may contain information such as comments and end punching. If a *RECORD* statement is not present in a DD, a default length of 80 will be used. When a *RECORD* statement exists, it must immediately follow the *TYPE* statement.

PROCESS dbspec

The *PROCESS* statement must be the next statement to appear in a DD. It indicates the set of data blocks that are to be created and loaded with DF data. The names of the data blocks may be included in the statement, or they may be submitted separately with the DF or by way of a user file:

- If the names are included in the *PROCESS* statement, they must be specified in the order in which the blocks are to be created and loaded, and they must be separated by commas. For example, *PROCESS BLOCK1,BLOCK2,BLOCK3*
- If the names are included in the DF, the form of the *PROCESS* statement must be either *PROCESS num* where *num* is a positive integer indicating the number of blocks to be loaded, or *PROCESS ALL* in which case all blocks named in the DF will be loaded.

. <u>If the names are included in a user file</u>, the form of the statement must be either *PROCESS num* in which case the first *num* blocks named in the *dbname* file (identified in the *USE* command) will be loaded; or *PROCESS ALL* in which case all blocks named in the *dbname* file will be loaded. The *dbname* file is a card image file, one block name per card image, Columns 1-8.

SKIP colspec

The *SKIP* statement causes either a new card image to be read from the DF file, or a specified number of columns of the current DF card image to be skipped (i.e., ignored). If the parameter *colspec* is omitted, a skip to the beginning of the next DF card image is performed; if *colspec* is a positive integer, that number of columns of the current card image will be skipped.

elname colspec

This statement must be used to load single-valued numeric, alphanumeric, and pointer data elements. The parameter *elname* must be a data element name as it appears in the block-type definition, and the parameter *colspec* must be a positive integer indicating the columnar width of the data value. For example, if the data element WEIGHT is a 7-character value for each block to be loaded, then the statement *WEIGHT 7* would appear in the DD.

NOTES: (1) If *elname = BN* the data field within the DF will contain the name of the data block (see explanation of the *PROCESS* statement).

(2) The *colspec* for a data element of type real may be specified *w.d.* as in the F-conversion format of FORTRAN; for example, the statement *WEIGHT 7.2* signifies a 7-character value having 2 digits to right of the decimal. This type of conversion specification would be used if the decimal were not in the DF. As in FORTRAN, if the decimal exists it overrides the *w.d.* specification.

(3) For data elements of type pointer, any SCOPE or CPFMS permanent file name--included in the DF as part of a cross-file pointer--must be parenthesized and positioned just after the block name; for example, BLOCKA(pfn,ID=id) and BLOCKB(level-3, level-4).

elname valspec,colspec

This statement must be used to load array data elements. The parameters *elname* and *colspec* are as defined in the *elname colspec* statement described earlier. The parameter *valspec* is either a positive integer, indicating the number of values comprising the array, or an asterisk (*) immediately followed by a positive integer, indicating the columnar width of that data field that specifies the number of values comprising the array. For example, if the element WEIGHTS is to contain six 8-character values for each block to be loaded, then the statement *WEIGHTS 6,8* would appear in the DD; but if WEIGHTS is a variable-length array for each block to be loaded, the statement

*WEIGHTS *3,8* would appear in the DD. In this instance, a three digit field within the DF will specify the number of values comprising WEIGHTS.

textelname colspec

This statement must be used to load data elements of the type text. The parameter *textelname* is the data element name as it appears in the block-type definition. The parameter *colspec* is either a positive integer, indicating the number of characters comprising the text, or an asterisk (*) immediately followed by a positive integer. This integer indicates the columnar width of that data field that specifies the number of characters comprising the text. For example, if the element EQUIP is to consist of 49 characters for each block to be loaded, then the statement *EQUIP 49* would appear in the DD; but if EQUIP is to consist of variable-length text for each block, then the statement *EQUIP *2* would appear in the DD. In this instance, a two-digit field in the DF will specify the number of characters of text for EQUIP.

rgname occspec

This statement must be used to specify the number of occurrences that are to be loaded for a repeating group; the statement must precede the DD statements that identify those elements which will be processed for the group in the current data base load. The parameter *rgname* is the repeating group name as it appears in the block-type definition. The parameter *occspec* is either a positive integer, indicating the number of

occurrences of *rgname* which are to be loaded for each block, or an asterisk (*) immediately followed by a positive integer, denoting the columnar width of the data field that specifies the number of occurrences comprising the repeating group.

ENDRG

The *ENDRG* statement must immediately follow the set of DD statements that identify those elements which will be processed for a repeating group in the current execution of the LOAD module. For every *rgname occspec* statement (see discussion just previous) appearing in a DD, there must exist an *ENDRG* statement.

Additional Information on Data Loading

When the value for a data element in the DF is found to be blank, no value will be loaded. In the case of an array, if the values specification (denoted in the DD) is n ($n>0$) but the number of non-blank values in the DF for that array is m ($m \leq n$) then only m non-blank values will be loaded for the block currently being processed. If an entire repeating group occurrence in the DF is blank, that occurrence is ignored.

These actions have two effects:

- The user wishing to load a value of 0 or 0.0 into an element (of type integer or real, respectively) cannot merely leave the appropriate field in the DF blank. He must provide the desired value.

. An element of the type alphanumeric, text, or pointer <u>cannot</u> be loaded (using the LOAD module) with a field of all blanks. If a user desires to achieve this result, he must use the UPDATE module after he has loaded the other values with the LOAD module.

Data Definition and Data File, an Example

The relationship between data to be loaded (i.e., the DF) and the data definition (DD) is best shown by way of an example. Hence, a coordinated sample, comprising a block type, DD and DF, has been provided. A brief discussion of the example follows a listing of the DF.

Block Type STATES

STATE	ALPHA	(Name of the state)	
YEARA	INTEGER	(Year of admission)	
CAPITAL	ALPHA	(Capital city)	
PRESPTR	POINTER ARRAY	(Points to blocks containing personal data regarding "native son" presidents)	
AREA	REAL	(Area in square miles)	
AREARANK	INTEGER	(Rank in area, 1-50)	
POPUL	INTEGER	(Population)	
POPRANK	INTEGER	(Rank in population, 1-50)	
ELECVOTE	INTEGER	(Number of electoral votes)	
CITY	ALPHA	(Name of city)	CITY and CITYPOP constitute the repeating group CITIES
CITYPOP	INTEGER	(Population of city)	

Sample Data Definition (DD)

```
TYPE STATES
RECORD 62
PROCESS 50
BN 8
SKIP
STATE 10
SKIP 5
YEARA 4
SKIP 5
CAPITAL 10
SKIP
PRESPTR *2,10
SKIP
AREA 10
SKIP 3
AREARANK 2
SKIP 6
POPUL 8
SKIP 3
POPRANK 2
SKIP 6
ELECVOTE 2
SKIP
CITIES *2
CITY 10
SKIP 2
CITYPOP 8
SKIP
ENDRG
```

Sample Data File (DF)

```
TEXAS
TEXAS           1845      AUSTIN
 2EISENHOW  JOHNSONL
 267339.45    2        11196730    4        26
10HOUSTON      1232802
DALLAS        844401
SANANTONIO    654153
FORT WORTH    393476
EL PASO       322261
AUSTIN        251808
CORP. CHR.    204525
LUBBOCK       149101
AMARILLO      127010
BEAUMONT      117548
VIRGINIA
VIRGINIA        1788      RICHMOND
 7WASHINGT  JEFFERSO  MADISON   MONROE    HARRISON  TYLER
WILSON
  40315.18   36       4648494   14        12
 7NORFOLK      307951
RICHMOND      249430
VA. BEACH     172106
NEWP. NEWS    138177
HAMPTON       120779
PORTSMOUTH    110963
ALEXANDRIA    110938
INDIANA
INDIANA         1816      INDIANAP.
 0
  36291.30   33       5193669   11        13
 6INDIANAP.    744743
FORT WAYNE    178021
GARY          175415
EVANSVILLE    138764
SOUTH BEND    125580
HAMMOND       107888
NH
N. H.           1788      CONCORD
 1PIERCE
   9304.89   44        737681   42         4
 0
WVA
W. VA.          1863      CONCORD
 0
  24181.73   41       1744237   34         6
 0
     .
     .
     .
```

In the example provided, the DF contains data for 50 blocks (but to conserve space only data for 5 of the 50 are shown). These data blocks, one for each of the 50 states (note that the statement *BN 8* in the DD indicates that block names are within the DF data), are variable in length because of the varying size of the array PRESPTR and the varying number of occurrences of the repeating group CITIES. For example, for the block TEXAS there are two values in the array PRESPTR; for VIRGINIA there are seven values; INDIANA has no values, as indicated by a zero in the *valspec* field; NH (New Hampshire) has one value for PRESPTR; and WVA (West Virginia), like INDIANA, has no values, again indicated by a 0 in the *valspec* field. For each block the number of values for PRESPTR is found in Columns 1-2 of the card image following the one containing data for the elements STATE, YEARA and CAPITAL. The number of occurrences of the repeating group CITIES is found, for each block, in Columns 1-2 of the card image following the one containing data for AREA, AREARANK, POPUL, POPRANK and ELECVOTES. Thus we see that TEXAS has ten occurrences of CITIES, VIRGINIA has seven, INDIANA has six, and NH and WVA have none, as indicated by a 0 in the *occspec* field.

3. The UPDATE Module

The UPDATE module allows the user to print, delete, create, modify and add data for the active data base file. Several

terms used in describing the individual commands of the UPDATE module need to be defined:

- Data block name list *(dbnamelist)* - The names, separated by commas, of one or more data blocks which are to be processed (for example, *BLOCKA*, *BLOCKB*, *BLOCKC*); a block type name bound in quotation marks (for example, "*SWBS*") signifying that all blocks of a given type are to be processed; or *ALL* indicating that all blocks in the data base file are to be processed. When the user issues a command in which a block type name or *ALL* is specified, he will receive a message indicating the number of blocks which will be processed; he will then be asked to verify or cancel his command.

- Data block item list (*dbitemlist*) - A subblock name, repeating group name, or data element name, or a combination of these names separated by commas; for example, *EL1,EL2,RG1,EL5*.

- Occurrence number list (*occnumberlist*) - Either an integer, signifying a repeating group occurrence number, or two integers separated by a hyphen, indicating a range of occurrence numbers, or a combination of these forms separated by commas; for example, *1,2,3-9,12,16*.

Commands of the UPDATE Module

PRINT dbnamelist/dbitemlist/occnumberlist

Examples:

PRINT BLOCKA	(Print the entire contents of the data block BLOCKA.)
PRINT BLOCKA,BLOCKB/EL1	(Print the data element EL1 of the data blocks BLOCKA and BLOCKB.)
PRINT BLOCKA/EL1,EL2,RG1	(Print all occurrences of the elements EL1 and EL2 and the repeating group RG1 of the data block BLOCKA.)
PRINT BLOCKA/EL5,EL6/3,4	(Print the third and fourth occurrences of the elements EL5 and EL6 of the block BLOCKA.)
PRINT BLOCKA/RG1/3-6,8	(Print the third, fourth, fifth, sixth, and eighth occurrences of all data elements in the repeating group RG1 of the block BLOCKA.)

PRINT "BT1"/EL5	(Print the data element EL5 of all data blocks of the block type BT1. Before the request will be processed, the user will have to respond to the question THERE ARE nnn DATA BLOCKS TO BE PROCESSED, SHALL WE CONTINUE?)
PRINT ALL	(Print the entire contents of every data block contained on this data base file. As in the previous example, printing will occur when an affirmative answer to the question is supplied.

*DELETE dbnamelist/dbitemlist/occnumberlist/**

To guard against accidental deletion of data, all *DELETE* commands must be verified before they are processed; that is, the user is asked if his intentions were properly expressed in his *DELETE* command. The optional asterisk (*) included at the end of the command signifies the user's desire to inspect the values of those items to be deleted prior to command verification; the asterisk serves the same purpose as consecutive *PRINT-DELETE* commands.

Examples:

DELETE BLOCKA	(Delete the data block BLOCKA.)
DELETE "BT1"	(Delete all data blocks of the type BT1.)
DELETE BLOCKA/EL1	(Delete the element EL1 from the data block BLOCKA; if EL1 is a member of a repeating group, all occurrences will be deleted.)
DELETE BLOCKA/RG1/3,4,6	(Delete the third, fourth, and sixth occurrences of the repeating group RG1 of the block BLOCKA.)
DELETE BLOCKA/EL5,EL6/6-9/*	(The values of the sixth through the ninth occurrences of EL5 and EL6 will be printed. The user will then be asked to verify the deletion.)

CREATE dbnamelist/btname

Examples:

CREATE BLOCKA/BT1	(Create a new data block of block type BT1 and name the block BLOCKA.)
CREATE BLOCKA,BLOCKB/BT1	(Create data blocks BLOCKA and BLOCKB of block type BT1.)

*MODIFY dbnamelist/dbitemlist/occnumberlist/**

When using the *MODIFY* command the user will be prompted for those data values he desires inserted into the data base.

Examples:

MODIFY BLOCKA/EL1	(The user will be prompted for a value or a set of values which will be used as the new contents of the data element EL1 in the block BLOCKA.)
MODIFY BLOCKA/RG1/3	(The third occurrence of the repeating group RG1 is to be modified. The user will be prompted for new values for each of the member elements in the repeating group. If a value for one of these elements is not to be modified, then the user must respond with the notation *$$* instead of a value. To illustrate, suppose RG1 consists of the elements AA, BB and CC. The user will be asked to provide a value for each of these elements. If the value of an element, say BB, is not to be modified, then the user must respond with *$$*.)
MODIFY BLOCKA/EL3,EL4,EL5/*	(The values of the three data elements will be printed, and the user will then be prompted for new values. If a value is not to be modified, the user merely enters the notation *$$* instead of a value. If one of the elements, say EL4, was a member of a repeating group, the user would be asked to provide a value for every occurrence of EL4 since no *occnumberlist* was specified.)

ADD dbnamelist/dbitemlist

The *ADD* command will add either a new occurrence to a repeating group, or a locally defined data element to a data block. Modifications to non-repeating group defined elements must be made by way of the *MODIFY* command, whether or not the data element has been loaded previously.

As with the *MODIFY* command the user of the *ADD* command will be prompted for those values he desires inserted into the data base.

Examples:

ADD BLOCKA/RG1	(Add a new occurrence to the repeating group RG1 of data block BLOCKA. The user will be prompted for element values in the manner described for the *MODIFY* command. If no value for one of these elements currently exists, the user must respond with the notation *$$*. To illustrate, suppose RG1 consists of the elements AA, BB and CC. The user will be asked to provide a value for each element. If the value for an element, say BB, does not exist, the user responds with *$$*.)
ADD BLOCKA/LOCEL1,LOCEL2	(Add the locally defined data elements LOCEL1 and LOCEL2 to block BLOCKA. The user will be prompted for the needed data values.)

4. The QUERY Module

The QUERY module allows the user to generate "hit" files (i.e., files containing names of data blocks stored on the active data base file(s) that satisfy certain user-specified criteria), and to print values retrieved from blocks named on the "hit" file.

Before specifying his "hit" file criteria, the user must already have activated the data base file(s) (as many as five

may be indicated) using the file management command *DATABASE*. Note that only while in the QUERY module may the user specify more than one file in the *DATABASE* command. Moreover, if the user has more than one data base file active at the time he exits the QUERY module to enter another module, all files will be "deactivated" and there will be no active file until one is specified in a subsequent *DATABASE* command.

Commands of the QUERY Module

The QUERY module *FOR* command allows a user to interrogate the active data base file(s) and to generate the initial QUERY hit file, or a new hit file that will replace the current hit file. The QUERY module *CHASE* command allows the user to traverse intra-file and cross-file pointers. Beginning with those data blocks named in the current hit file, the traversal process may progress through up to eight different pointer elements resulting in a new hit file. The *QUALIFY* command allows a user to generate a new hit file that is a subset of the current hit file.

The QUERY module also supports several auxiliary commands, *SAVE*, *UNSAVE*, *HITS*, *BATCH* and *ENDBATCH*, which allow a user to manipulate hit files and to store QUERY commands on a queue for subsequent execution.

The QUERY module *PRINT* command allows a user to retrieve values from data blocks named in the current hit file and to print these values in one of three report formats: columnar, row and simple list.

QUERY commands may be three lines long (a line may contain as many as 80 characters). An ampersand (&) entered as the last non-blank character of a line, indicates a continuation.

FOR hitfilecriteria

The *FOR* command initiates generation of a hit file using information contained in the active data base file(s). Note that this is the only command that can generate the initial hit file. If a hit file already exists (resulting from a previous *FOR*, *CHASE* or *QUALIFY* command) and it has not been saved (see the discussion of the *SAVE* command), it will be unloaded when the new hit file is generated.

Acceptable *hitfilecriteria* for the *FOR* command are as follows:

(i) A relational expression, written either as

elname relational operator value
in which case the relational operator must be .*EQ*.,.*NE*., .*GE*.,.*GT*.,.*LE*., *or* .*LT*.;

or *elname* .*BET*. *value1*, *value2*
in which case the operator .*BET*. indicates that only values within the range defined by *value1* and *value2* inclusive will be considered.
Note that in a relational expression the parameter *elname* must be the name of an inverted data element, and that whenever a value is alphanumeric it must be bound in quote marks (").

(ii) A block type expression written as

BT .*EQ*. *btname*
where *btname* is the name of a block type stored on the active data base file(s).

(iii) The name of a saved hit file.

(iv) A list of data block names separated by commas. The list must begin and end with a slash; for example, */BLOCKA, BLOCKB, BLOCKC/*

(v) Any combination of expressions (i) through (iv) connected by the Boolean operators *.AND.* or *.OR.* Expressions may be enclosed in parentheses when it is necessary to alter the normal *.AND.* before *.OR.* precedence.

Examples:

```
FOR HEIGHT .GT. "6FT."

FOR /LINCOLN, KENNEDY/

FOR HEIGHT .GT. "6FT." .AND. FILEX .OR. /LINCOLN/

FOR SURNAME .BET. "ADAMS", "LINCOLN" .AND. YEARB .GT.
1799
```

QUALIFY hitfilecriteria

The *QUALIFY* command allows a user to generate a new hit file that is a subset of the current hit file (i.e., the file resulting from a previous *FOR*, *CHASE* or *QUALIFY* command).

If the user specifies a search on a non-inverted element, he is given a message specifying the number of data block accesses that must be made in order to satisfy the non-inverted search, and he is asked to verify whether he wishes to continue the search procedure. The user should be aware that non-inverted searches may be quite costly and he should therefore act accordingly.

Acceptable *hitfilecriteria* for the *QUALIFY* command are as follows:

(i) A relational expression as described for the *FOR* command except that *elname* can be the name of any single-valued, non-inverted data element.

(ii) Any combination of relational expressions connected by the Boolean operators *.AND.* or *.OR.* The normal precedence of *.AND.* before *.OR.* may be altered by enclosing expressions in parentheses.

Examples:

FOR HEIGHT .GT. "6FT." .AND. FILEX	(The user generates a hit file.)
QUALIFY YEARB .BET. 1700,1896	(The user qualifies the hit file generated by the above *FOR* command.)
QUALIFY INITIAL .EQ. "Q."	(The hit file generated by the *QUALIFY* command is qualified.)

CHASE ptrspec

The *CHASE* command allows a user to traverse intra-file and cross-file pointers. Beginning with those data blocks in the current hit file, the traversal process may progress through up to eight different pointer elements and will result in a new hit file. The parameter *ptrspec* is a list of up to eight pointers separated by slashes; for example, *PTR1/PTR2/PTR3.*

If a cross-file pointer is encountered, the file pointed out will also be processed providing one of the following three conditions is satisfied:

- It has been attached with the file management command *ATTACH.*
- It has not been attached and does not have a password.
- It has not been attached and the user can supply the appropriate password.

In the first two cases the processing will continue without interruption. In the third case, the user will be prompted for the password.

Examples:

CHASE STATEPTR	(Consider a data base containing information about the U.S. presidents. Consider also that block types ELECTION (containing a pointer PRESPTR), PRES (containing a pointer STATEPTR) and STATE are stored on the data base. If the current hit file contains two data blocks, JACKSON and MADISON, then this command will generate a hit file containing the block names SC and VIRGINIA.)
CHASE PRESPTR/STATEPTR	(If for the presidents data base the current hit file contains the block names E1960, E1964, E1968, E1972 and E1976 (data blocks containing information relative to the presidential elections for the years 1960-76), then this command will generate a hit file containing the names MASS, TEXAS, CAL and GEORGIA.)

SAVE lfn

The *SAVE* command allows a user to assign a name to the current hit file and to save the file for subsequent processing; the parameter *lfn* is the user-supplied file name. A saved hit file can be referenced in a *FOR* command at any time. The user should note that saved hit files (and the current hit file for that matter) can become meaningless if the data base file(s) is unloaded (via the *UNLOAD* command) even though the hit files may still exist.

UNSAVE lfn

The *UNSAVE* command unloads the specified saved hit file *(lfn)* and frees the name for reuse in a subsequent *SAVE* command.

HITS lfn

The *HITS* command returns the number of hits (i.e., block names) on a specified saved hit file *(lfn)*, or on the current hit file when the parameter *lfn* is omitted from the command.

BATCH

The *BATCH* command allows a user to submit QUERY commands and to have them checked for syntax and stored on a queue for subsequent execution. This feature enables the user to leave his terminal while a set of commands is executing. After a *BATCH* command has been issued, each succeeding command will be checked for syntactical validity (diagnostics are printed as necessary) and will then be stored on a command queue.

ENDBATCH

The *ENDBATCH* command causes the QUERY module to execute in sequence those commands which have accumulated on the command queue as a result of the *BATCH* command (see above). When the last command on the queue has been executed, QUERY waits for the user to enter another command.

PRINT printspec

The *PRINT* command allows a user to retrieve values from data blocks named in the current hit file and to have these values printed either on his terminal, on a report file, or at both destinations (see the discussion of the auxiliary command *OUTPUT)*. In addition, the *PRINT* command allows the user to

perform simple arithmetic operations; to sort, average and sum; and to specify an output format of either columnar, row, or simple list.

The parameter *printspec* may consist of up to three items written as

format elnamelist sortspec

where the item *elnamelist* identifies those entities which are to be printed, and *format* and *sortspec* denote a report format and sort specification, respectively. A more detailed description of the three *printspec* items follows:

. *format* - If the report format is to be a simple list, then *format* must be omitted from the *printspec*. If a columnar report format is required, *format* must be the designation *IN COLS* in which case data values will be printed in columns (up to five for the user's terminal; eight for the report file) and those data element names specified in the *elnamelist* will appear as column headings. If a row format is required, *format* must be the designation *IN ROWS* in which case values will be paired with their data element names (from *elnamelist*) and printed in row fashion (up to three pairs per line for the terminal; five pairs for the report file).

. *elnamelist* - The following entities may appear in an *elnamelist*: *DB* (denoting data base name), *BT* (denoting block type name), *BN* (denoting data block), data element names and repeating group names as they appear in the block-type definition, and parenthesized arithmetic operations involving two valid data elements. (The two data elements of an arithmetic operation must be of the same block type and must be single-valued and of type numeric; if repeating group elements, they must be members of the same repeating group.) An arithmetic operation is written as one of the following:

(elname1 + elname2)	Sum of *elname1* and *elname2*
(elname1 - elname2)	Difference
*(elname1 * elname2)*	Product
(elname1 / elname2)	Quotient

where *elname1* and *elname2* are valid data element names; the names need not appear as individual data elements in

the *elnamelist*. If the sum and/or average of any single-valued, numeric element is to be calculated, the *elnamelist* may include the designations *SUM(elname)* and/or *AVE(elname)*. Entities appearing in an *elnamelist* must be separated by commas. When *IN COLS* is specified in the *format*, the *elnamelist* may include only *DB*, *BT*, *BN* and a set of data element names; the data elements must be single-valued and must either be non-repeating group elements, or members of the same repeating group.

. *sortspec* - The optional parameter *sortspec* is valid only when *IN COLS* is specified in the *format*. The *sortspec* identifies the ascending (i.e., *HIGH*) or descending *(LOW)* order in which a set of data elements will be sorted; those elements named in the *sortspec* must also be named in the *elnamelist*. A *sortspec* must be written as

$$\mathit{SORTED\ ON} \begin{bmatrix} \mathit{HIGH} \\ \mathit{LOW} \end{bmatrix} \mathit{elname1} \begin{bmatrix} ,\mathit{HIGH} \\ \mathit{LOW} \end{bmatrix} \mathit{elname2}\ldots$$

where *elname1* and *elname2* are element names. When more than one sort is specified, the first (i.e., *elname1*) is the most significant. The brackets ([]) indicate that the user must make a choice between the designations *HIGH* and *LOW*; for example, *SORTED ON HIGH AA, LOW BB*.

Examples:

(1) The command PRINT IN COLS WEIGHT, LENGTH, COST will result in the following arrangement:

WEIGHT	LENGTH	COST
300	400	500
60	70	800
700	60	90

(2) The command PRINT IN ROWS LENGTH, WEIGHT, COST, DEPTH, AREA will result in the following arrangement:

```
LENGTH:  400  WEIGHT:  300  COST:  500
DEPTH:   400  AREA:    600

LENGTH:   70  WEIGHT:   60  COST:  800
DEPTH:   100  AREA:    400
```

(3) The command PRINT WEIGHT, LENGTH, COST will result in the following simple listing:

WEIGHT	300
LENGTH	400
COST	500
WEIGHT	60
LENGTH	70
COST	800

5. The STATS Module

The STATS module furnishes the user with statistics about the various data blocks, block types, data definitions and inverted data elements (i.e., the keys) stored on the active data base. After he enters the STATS module, the user is informed as to how many data blocks, block types, data definitions and keys are included on the data base; the user may then request more detailed information related to each of these items.

Commands of the STATS Module

Eight commands, *DBNAMES*, *DBDUMP*, *BTNAMES*, *DDNAMES*, *KEYNAMES*, *KEYRANGES*, *KEYDUMP* and *DATE*, are supported by the STATS module.

DBNAMES btnamelist

The *DBNAMES* command allows the user to print the names of the data blocks of a subset of the active data base. The parameter *btnamelist* consists either of one or more block type names, separated by commas; or of the word *ALL*,

indicating that the names of all data blocks on the active data base file are to be printed.

Example:

DBNAMES PRES, ELECTION	(Print the names of all blocks of the block type PRES and the block type ELECTION.)

DBDUMP dumpspec

The *DBDUMP* command allows the user to obtain an octal printout of the data blocks for a specified subset of the active data base. The parameter *dumpspec* must be either one or more block names, separated by commas; or the names, each enclosed in quote marks and separated by commas, of one or more block-type definitions; or *ALL*, indicating that all blocks on the data base file are to be printed.

Examples:

DBDUMP LINCOLN	(Print the contents of the data block named LINCOLN.)
DBDUMP "PRES", "ELECTION"	(Print the contents of all blocks of block types PRES and ELECTION.)

BTNAMES

The *BTNAMES* command allows the user to print the names of the block-type definitions stored on the active data base; the number of data blocks is printed adjacent to each block type name.

DDNAMES

The *DDNAMES* command allows the user to print the names of the data definitions stored on the active data base.

KEYNAMES

The *KEYNAMES* command allows the user to print the names of the inverted data elements (or keys) stored on the active data base.

KEYRANGES keynamelist

The *KEYRANGES* command allows the user to print the number of values, the lowest value and the highest value, for a subset of the keys stored on the active data base. The parameter *keynamelist* must be either one or more names, separated by commas, of the keys whose ranges are to be printed; or *ALL* indicating that the ranges of all keys are to be printed.

KEYDUMP keynamelist

The *KEYDUMP* command allows the user to print all values for a subset of the keys stored on the active data base. The parameter *keynamelist* must be either one or more names, separated by commas, of the keys whose values are to be provided, or *ALL*, indicating that the values of all keys are to be printed.

DATE

The *DATE* command allows the user to print the date and time at which the active data base file was created, and the date and time at which the file was last compressed.

6. The DUMP Module

The DUMP module allows the user to dump the contents of data blocks onto a card image file (DF) formatted as described by a data definition (DD). The DUMP module provides the following features:

- The user can save DD's. A saved DD can be used at a later time to dump another set of blocks and/or create and load a set of blocks. (See the description of the LOAD module for a discussion of the data base load process.) A saved DD can be printed on request.
- The names of the data blocks to be dumped can be specified in the DD or on a user-supplied file.
- The amount of space allotted in the DF for repeating groups and arrays can vary from block to block depending on the number of occurrences in the repeating groups and the number of values in the arrays.

Commands of the DUMP Module

USE ddname, dfname [,*dbname*]

The *USE* command initiates the data base dump and identifies those files which are to be used in the process. The parameter *ddname* is either the name of a DD already stored on the data base, or the name of a local file containing a DD. Other parameters include *dfname*, the name of the local DF file, and *dbname* (optional), the name of the local file that contains the names of those blocks to be dumped.

SAVE ddname

The *SAVE* command causes a DD, existing on a local file identified in the most recent *USE* command, to be stored on the data base so it may be invoked in a subsequent data base

dump (i.e., subsequent *USE* command). The parameter *ddname* must be a 1- to 8-character alphanumeric string beginning with a letter.

UNSAVE ddname

The *UNSAVE* command causes a previously saved DD to be removed from the data base.

PRINT ddname

The *PRINT* command causes a previously saved DD to be printed.

Data Definition Statements

TYPE btname

The *TYPE* statement indicates that data blocks to be dumped according to the current DD are of block type *btname*. This statement must precede all other statements of a DD.

RECORD length

The *RECORD* statement, in which the parameter *length* is a positive integer less than or equal to 80, indicates the length of the data string in each of the DF card images. Thus the statement *RECORD 50* would indicate that the data being dumped is to be contained in Columns 1-50 of the DF card images. If the *RECORD* statement is omitted from a DD, a default length of 80 will be used. When a *RECORD* statement exists, it must immediately follow the *TYPE* statement.

PROCESS dbspec

The *PROCESS* statement must be included in a DD. It indicates the set of data blocks whose values are to be dumped to the DF. The names of the data blocks may be included in the statement, or they may be submitted separately by way of a user file. The user may request the dumping of all blocks of the type specified in the *TYPE* statement.

. If the names are included in the *PROCESS* statement, they must be specified in the order in which the blocks are to be dumped, and they must be separated by commas. For example, *PROCESS BLOCK1,BLOCK2,BLOCK3*.

. If the names are included in a user file, the form of the statement must be either *PROCESS num* in which case the first *num* blocks named on the *dbname* file (identified in the *USE* command) will be dumped, or *PROCESS ALL* in which case all blocks named on the *dbname* file will be dumped. The *dbname* file is a card image file, one block name per card image, columns 1-8.

. If all blocks of a certain type (as indicated in the *TYPE* statement) are to be dumped, the form of the statement must be *PROCESS ALL* and no *dbname* file need be specified in the *USE* command.

SKIP colspec

The *SKIP* statement either causes the current card image to be written to the DF file, or causes a specified number of columns of the current DF card image to be skipped (i.e., ignored). If the parameter *colspec* is omitted, the card image

built in memory will be written to the DF; if *colspec* is a positive integer, that number of columns in the card image buffer will be skipped.

elname colspec

This statement must be used to dump single-valued numeric, alphanumeric, and pointer data elements. The parameter *elname* must be a data element name as it appears in the block-type definition, and the parameter *colspec* must be a positive integer indicating the columnar width of the data value to be dumped.

NOTES: (1) If *elname* = *BN*, the data field within the DF will contain the name of the data block being dumped.

(2) Data values of type real will be dumped in either floating point format or exponential format, depending upon which provides the greater accuracy.

(3) If a data element of type pointer consists of a cross-file pointer value, the SCOPE or CPFMS permanent file name will be included in parentheses in the DF immediately after the block name: For example, BLOCKA (PERMFILE, ID=ABCD) for a SCOPE file; and BLOCKB (DDG120, DDG120) for a CPFMS file.

elname valspec, colspec

This statement must be used when array data elements are to be dumped. The parameters *elname* and *colspec* are as defined in the *elname colspec* statement (see previous description). The parameter *valspec* is either a positive integer, indicating the number of values to be dumped from the array, or an asterisk (*) immediately followed by a positive integer, indicating the columnar width of the field in the data that is to receive the number of values that will be dumped for the array.

textelname colspec

This statement must be used when textual data elements are to be dumped. The parameter *textelname* is the data element name as it appears in the block-type definition. The parameter *colspec* is either a positive integer, indicating the number of text characters to be dumped, or an asterisk (*) immediately followed by a positive integer. This integer indicates the columnar width of the field in the data that is to receive the number of text characters to be dumped; the text string will be dumped immediately following this field.

rgname occspec

This statement must be used to specify the number of occurrences that are to be dumped from a repeating group; the statement must precede the DD statements that identify those elements which will be processed for the repeating group in the current data base dump. The parameter *rgname* is the repeating

group name as it appears in the block-type definition. The parameter *occspec* is either a positive integer, indicating the number of occurrences of *rgname* which are to be dumped for each block, or an asterisk (*) immediately followed by a positive integer, denoting the columnar width of the field in the data that is to receive the number of occurrences that will be dumped from the repeating group.

ENDRG

The *ENDRG* statement must immediately follow the set of DD statements that identify those elements which will be processed for a repeating group in the current execution of the DUMP module. For each *rgname occspec* statement (see previous paragraph) of a DD, there must exist an *ENDRG* statement.

Additional Information on Data Dumping

When the value for a data element to be dumped exceeds its *colspec* specification in the DD, the corresponding field in the DF will be filled with asterisks (*'s). When the number of values in an array (or the number of repeating group occurrences) is greater than its *valspec* specification in the DD, excess values in the array (or repeating group occurrences) will not be dumped but an appropriate message will be printed; if the number is less than the *valspec*, excess fields in the DF will be blank-filled and a message will be printed.

Finally, when no value has been loaded for an element that is to be dumped, the corresponding field in the DF will be blank-filled and a message will be printed.

Sample Data Definition and Data File

The relationship between the values to be dumped to the data file (DF) and the data definition (DD) is best shown by way of an example. Hence, a coordinated sample comprising a DD and a DF ensues, followed by a brief discussion of the sample. Assume data elements in the DD (i.e., STATE, YEARA, etc.) are as defined in the block type STATES which was provided in the earlier discussion of the LOAD module.

Sample Data Definition (DD)

```
TYPE STATES
PROCESS ARIZONA,OHIO,VERMONT,RI
STATE 10
SKIP 5
YEARA 4
SKIP 5
CAPITAL 10
SKIP
PRESPTR *2,10
SKIP
AREA 7
SKIP 3
AREARANK 2
SKIP 6
POPUL 8
SKIP 3
POPRANK 2
SKIP 6
ELECVOTE 2
SKIP
CITIES *2
CITY 10
SKIP 2
CITYPOP 8
SKIP
ENDRG
```

Sample Data File (DF)

```
ARIZONA         1912    PHOENIX
 0
13909.     6         1772482  33       6
 2PHOENIX      581562
TUCSON       262933
OHIO            1803    COLUMBUS
 7GARFIELD  GRANT    HARDING   HARRISON  HAYES   MCKINLEY  TAFT
41222.5   35       10652017   6       25
 9CLEVELAND    750879
COLUMBUS     540025
CINCINNATI   452524
TOLEDO       383818
AKRON        275425
DAYTON       243601
YOUNGSTOWN   139788
CANTON       110053
PARMA        100261
VERMONT         1791     MONTPELIER
 1COOLIDGE
9609.38   43         444732   49       3
 0
R.I.            1790     PROVIDENCE
 0
1214.83   50         949723   29       4
 1PROVIDENCE     179116
```

In this example, values of four data blocks (ARIZONA, OHIO, VERMONT and RI) of the block type STATES have been dumped to the DF. The first card image for each block contains values for the data elements STATE, YEARA, and CAPITAL (e.g., Vermont, 1791, Montpelier). The second card image contains values for the array PRESPTR; preceding the values is a two-digit field containing the number of values in PRESPTR (e.g., OHIO has the seven values Garfield, Grant, Harding, Harrison, Hayes, McKinley and Taft, whereas VERMONT has only the value Coolidge). Since neither ARIZONA nor RI have had native-son Presidents, their second card images contains zero in columns 1-2.

The third card image of each block contains values for the elements AREA, AREARANK, POPUL, POPRANK and ELECVOTE. Values on the fourth card image and those that follow for the repeating group CITIES are dumped, one occurrence per card image. (In effect, each card image contains values for the elements CITY and CITYPOP.) Preceding the values for the first occurrence is a two-digit field indicating the number of occurrences of CITIES. Thus, for the block OHIO there are nine occurrences of CITIES (designated by the integer 9 adjacent to Cleveland); the blocks ARIZONA and RI have two and one, respectively. The fourth card image for the block VERMONT, which has no CITIES data, contains zero in columns 1-2.

ACKNOWLEDGMENTS

The authors wish to acknowledge Thomas R. Rhodes of DTNSRDC and Roger Berry of the Naval Ship Engineering Center (NAVSEC) for their technical suggestions and criticisms. The authors also wish to recognize Dr. Peter R. Bono, formerly of NAVSEC, for his contribution to the design of the CDMS QUERY module, and Maria Kadala of DTNSRDC, for her computer programming assistance.

APPENDIX A

COMRADE PERMANENT FILE MANAGEMENT SYSTEM

The COMRADE Permanent File Management System (CPFMS) affords the COMRADE user/programmer a special file management capability not otherwise available under the SCOPE permanent file system. Using CPFMS, the user/programmer is able to limit file access to those persons associated with a particular design project, only. This control is achieved by creating CPFMS files which may only be accessed from within a COMRADE application system.

FILE-NAMING CONVENTION

Each CPFMS file is assigned a unique four-level designation. The L1 (first-level) name COMR is given to all CPFMS files. The L2 (second-level) name is that of the application system with which the file is associated (for example, ISDS). The L3 (third-level) name is that of a project within the application system with which the file is associated (for example, DDG3). The L4 (fourth-level) name may be freely chosen; it is used to suggest the contents of the file (WEIGHTS, for example). The L2, L3, and L4 names are concatenated, using dashes, to form a unique permanent file name which can be processed by the SCOPE permanent file system. Each level name may be composed of as many as eight characters. The designation ISDS-DDG3-USERS (with ID=CPFM) typifies the structure of a CPFMS file name.

FILE ACCESS CONTROL

CPFMS files may only be used within the application system for which they were created; thus only the L3 and L4 names actually need to be submitted by the would-be user. Two schemes for controlling access to CPFMS files from within application systems are provided--the pseudo-password, and the File-Access-Key/File-Access-Lock (FAK/FAL). Both are assigned at the time the file is cataloged within the CPFMS system. If the pseudo-password is used, access to that file will require that the user submit a pseudo-password for a check. If the password submitted by the user is identical to the one assigned at the time the file was cataloged, access is granted.

If the FAK/FAL is used, the operation is more complex. The FAL assigned to a file consists of a 20-bit field, each bit associated with a different aspect of a project effort (for example, the hull design portion). The FAK assigned to each COMRADE user consists of 60 bits, organized into three 20-bit fields called sub-FAK's. Each of the three sub-FAK's pertains to a different level of file access: (1) Attach with read permission only; (2) attach with read-write permission; or (3) attach with purge permission. Each bit of the 20-bit sub-FAK may itself be associated with a particular subset of the project effort. Those sub-FAK bit positions that correspond to the subsets that the user is allowed to use will contain 1's. Each of the 20-bit sub-FAK's will be

compared with the FAL of the file in question to determine which, if any, of these levels of file-access will be permitted that particular user. The FAK/FAL check will be considered a success (that is, access will be granted) if the bits in the user's 20-bit sub-FAK that correspond to the 1-bits contained in the FAL are 1's.

APPENDIX B

DTNSRDC CHARACTER SET

DISPLAY CODE	CHARACTER	PUNCH 026	PUNCH 029 IF DIFF	7-TRACK TAPE	NOTE NAME
01	AAA	12-1		61	
02	BBB	12-2		62	
03	CCC	12-3		63	
04	DDD	12-4		64	
05	EEE	12-5		65	
06	FFF	12-6		66	
07	GGG	12-7		67	
10	HHH	12-8		70	
11	III	12-9		71	
12	JJJ	11-1		41	
13	KKK	11-2		42	
14	LLL	11-3		43	
15	MMM	11-4		44	
16	NNN	11-5		45	
17	OOO	11-6		46	
20	PPP	11-7		47	
21	QQQ	11-8		50	
22	RRR	11-9		51	
23	SSS	0-2		22	
24	TTT	0-3		23	
25	UUU	0-4		24	
26	VVV	0-5		25	
27	WWW	0-6		26	
30	XXX	0-7		27	
31	YYY	0-8		30	
32	ZZZ	0-9		31	
33	000	0		12	
34	111	1		01	
35	222	2		02	
36	333	3		03	
37	444	4		04	
40	555	5		05	
41	666	6		06	
42	777	7		07	
43	888	8		10	
44	999	9		11	
45	+++	12	12-8-6	60	PLUS
46	---	11		40	MINUS
47	***	11-8-4		54	ASTERISK
50	///	0-1		21	SLASH
51	(((	0-8-4	12-8-5	34	LEFT PAR
52	)))	12-8-4	11-8-5	74	RT PAR

DISPLAY CODE	CHARACTER	PUNCH 026	PUNCH 029 IF DIFF	7-TRACK TAPE	NOTE NAME
53	$$$	11-8-3		53	DOLLAR
54	===	8-3	8-6	13	EQUAL
55				20	BLANK
56	,,,	0-8-3		33	COMMA
57	...	12-8-3		73	PERIOD
60	###	0-8-6	8-3	36	POUND
61	(((	8-7	12-8-2	17	L BRACKET
62	)))	0-8-2	11-8-2	32	R BRACKET
63	:::	8-2			COLON
64	"""	8-4	8-7	14	QUOTE
65	___	0-8-5		35	UNDERLINE
66	!!!	11-8-2	12-8-7	52	EXCLAMATION
66	!!!	11-0		52	EXCLAMATION
67	&&&	0-8-7	12	37	AMPERSAND
70	'''	11-8-5	8-5	55	APOSTROPHE
71	???	11-8-6	0-8-7	56	QUESTION
72	<<<	12-8-2	12-8-4	72	LESS THAN
72	<<<	12-0		72	LESS THAN
73	>>>	11-8-7	0-8-6	57	GREATER
74	@@@	8-5	8-4	15	AT
75	\\\	12-8-5	0-8-2	75	REVERSE SLANT
76	^^^	12-8-6	11-8-7	76	CIRCUMFLEX
77	;;;	12-8-7	11-8-6	77	SEMICOLON
55		8-6	0-8-4		BLANK

APPENDIX C

SAMPLE TERMINAL SESSION USING CDMS

DIALOGUE AT THE TERMINAL

(The numbers in parentheses at the right have been provided to facilitate the discussion that follows).

```
NSRDC 6700 INTERCOM V4.4
DATE  06/04/76
TIME  12.23.10.
LOGIN,CXXXYYYZZZ,1234567890,SUP                                   (1)

COMMAND- COMRADE, CDMS                                            (2)

COMRADE       TIME:     12.24.16.
              DATE:     06/04/76

REQUEST- DEFINE                                                   (3)

REQUEST- ATTACH,BTDATA,PRESIDENTSBLOCKTYPE,ID=CABG.               (4)

REQUEST- DATABASE,NEWDB                                           (5)

REQUEST- CREATE PRES/BTDATA                                       (6)

REQUEST- PRINT PRES                                               (7)


BLOCK TYPE-PRES

SUB-BLOCK  1-PERSONAL

ELEMENT    1- SURNAME     ALPHA      INVERTED
ELEMENT    2- FIRSTNAM    ALPHA
ELEMENT    3- INITIAL     ALPHA
ELEMENT    4- STATEB      ALPHA      INVERTED
ELEMENT    5- YEARB       INTEGER    INVERTED
ELEMENT    6- MONTHB      ALPHA
ELEMENT    7- DAYB        INTEGER
ELEMENT    8- PARTY       ALPHA      INVERTED
ELEMENT    9- RELIGION    ALPHA      INVERTED
ELEMENT   10- ANCESTRY    ALPHA
ELEMENT   11- COLLEGE     ALPHA
```

```
SUB-BLOCK  2- FAMILY

ELEMENT    1- WIFE          ALPHA
ELEMENT    2- YEARM         INTEGER
ELEMENT    3- CHILDREN      INTEGER

REPEATING GROUP  1- MARRIAGE
                   ELEMENT    1
                   ELEMENT    2
                   ELEMENT    3
```

REQUEST- CATALOG,NEWDB,USPRESDATABASE,ID=CABG,AC=1234. (8)

REQUEST- LOAD (9)

REQUEST- EDITOR (10)

TYPE "EDITOR" UPON ENTERING COMMAND MODE. AFTER SAVING FILE(S) AND LEAVING EDITOR, TYPE "CDMS" TO RETURN TO THIS MODULE.

COMMAND- EDITOR (11)

```
..C S
ENTER LINES
TYPE PRES
RECORD 80
PROCESS ALL
BN 8
SKIP 2
SURNAME 10
SKIP 2
FIRSTNAM 10
SKIP 2
INITIAL 1
SKIP 2
STATEB 10
SKIP
YEARB 4
SKIP 2
MONTHB 10
SKIP 2
DAYB 2
SKIP 2
PARTY 10
SKIP 2
RELIGION 10
SKIP 2
ANCESTRY 10
SKIP 2
```

```
COLLEGE 10
SKIP
MARRIAGE *1
WIFE 10
SKIP 2
YEARM 4
SKIP 2
CHILDREN 2
SKIP
ENDRG
=
..S PRESLID N
..B

COMMAND- CDMS                                                    (12)

REQUEST- ATTACH,DATFILE,PRESIDENTSDATA,ID=CARG.                  (13)

REQUEST- USE PRESLID,DATFILE                                     (14)

   37 DATA BLOCKS CREATED
        LOAD COMPLETE

REQUEST- FILES                                                   (15)

  *BTDATA       *NEWDB       PRESLID       *DATFILE

REQUEST- UNLOAD,BTDATA,PRESLID,DATFILE.                          (16)

REQUEST- QUERY                                                   (17)

REQUEST- FOR PARTY .EQ. "REPUBLICAN"                             (18)

REQUEST- HITS                                                    (19)

   THERE ARE      17 HITS ON HITFILE

REQUEST- PRINT SURNAME                                           (20)

SURNAME     LINCOLN
SURNAME     JOHNSON
SURNAME     GRANT
SURNAME     HAYES
SURNAME     GARFIELD
SURNAME     ARTHUR
SURNAME     HARRISON
SURNAME     MCKINLEY
SURNAME     ROOSEVELT
SURNAME     TAFT
SURNAME     WILSON
SURNAME     HARDING
SURNAME     COOLIDGE
```

```
SURNAME      HOOVER
SURNAME      EISENHOWER
SURNAME      NIXON
SURNAME      FORD

REQUEST- QUALIFY STATEB .EQ. "OHIO"                              (21)

REQUEST- PRINT IN COLS SURNAME, FIRSTNAM, YEARB,
         RELIGION                                                (22)
```

SURNAME	FIRSTNAM	YEARB	RELIGION
GARFIELD	JAMES	1831	DIS.CHRIST
GRANT	ULYSSES	1822	METHODIST
HARDING	WARREN	1865	BAPTIST
HARRISON	BENJAMIN	1833	PRESBYT.
HAYES	RUTHEFORD	1843	METHODIST
MCKINLEY	WILLIAM	1843	METHODIST
TAFT	WILLIAM	1857	UNITARIAN

```
REQUEST- UPDATE                                                  (23)

REQUEST- MODIFY LINCOLN,WILSON/PARTY/*                           (24)

DATA BLOCK = LINCOLN

PARTY        REPUBLICAN

ENTER VALUE- $$                                                  (25)

DATA BLOCK = WILSON

PARTY        REPUBLICAN

ENTER VALUE- DEMOCRATIC                                          (26)

REQUEST- PRINT WILSON                                            (27)

DATA BLOCK = WILSON

SURNAME      WILSON
FIRSTNAM     WOODROW
INITIAL
STATEB       VIRGINIA
YEARB            1856
MONTHB       DECEMBER
DAYB               28
PARTY        DEMOCRATIC
RELIGION     PRESBYT.
ANCESTRY     ENGLISH
COLLEGE      PRINCETON
```

```
*** REPEATING GROUP MARRIAGE ***
WIFE          ELLEN                              OCC-1
YEARM
CHILDREN         3
WIFE          EDITH                              OCC-2
YEARM          1914
CHILDREN         0
*** END REPEATING GROUP ***

REQUEST- OUTPUT,FILE                                            (28)

REQUEST- PRINT ALL                                              (29)

THERE ARE    37 DATA BLOCKS TO BE PROCESSED.
SHALL WE CONTINUE?- YES                                         (30)

REQUEST- SEND,6700                                              (31)

REQUEST- EXIT                                                   (32)

???? LOGOUT                                                     (33)

CPA       54.935 SEC
CPB         .000 SEC
SS        55.970 SEC
EST. SYSTEM COST  $     8.82
EST. CONNECT COST $     9.27
CONNECT TIME      0 HRS.   54 MIN.
 06/04/76  LOGGED OUT AT 13.17.14.
```

DISCUSSION

The user "logs in" to INTERCOM (1) and enters the CDMS procedure (2). He wishes to create a data base containing information pertaining to the United States Presidents. First, he enters the DEFINE module (3) for the purpose of creating a block type. The input to the DEFINE module, i.e., the block type definition, has been stored previously on a SCOPE permanent file which the user now attaches (4). He indicates that the local file name of the active data base (in this case, the data base he is about to create) is NEWDB (5), and

he initiates the block type definition (6). The newly-defined block type, PRES, is printed (7) and the new data base, NEWDB, is made permanent (8).

To load data blocks onto the data base, the user must leave the DEFINE module and enter the LOAD module (9). Data items to be loaded onto the data base exist, although the definition of this data, the DD, does not. The user wishes to create this DD using EDITOR (10). He enters the EDITOR program (11), creates and saves a file (the DD), and reenters the LOAD module (12). The actual information describing the U.S. Presidents exists on a permanent file which the user attaches (13) and the load is initiated by the USE command (14).

After the data base has been successfully loaded, the user determines the names of those local files that are still active (15) and then disposes of those he no longer needs (16).

He now enters the QUERY module (17). His first query (18) produces 17 "hits" (19), indicating that there have been 17 Republican presidents. He prints the SURNAME of these presidents (20). Wishing to know more about the Republican presidents who were born in Ohio, he enters a QUALIFY command (21) and receives a report on the seven G.O.P. Buckeye presidents, the information printed in columns (22).

Noting that some of the information regarding PARTY seems to be in error, the user switches to the UPDATE module (23). Specifically, he intends to modify the PARTY of those data blocks named LINCOLN and WILSON (24). Remembering at the

last moment, however, that Lincoln was indeed a Republican, the user cancels this part of the Update request (25) and proceeds to correct PARTY WILSON (26). He now requests that the entire contents of the data block WILSON be printed (27).

Finally, the user, wishing to have a complete printout of his data base contents, switches to the off-line mode (which will result in his output information being written to the Report File rather than to the user's terminal) (28), and then issues the request to print all data blocks (29). After the request has been affirmed (30), the data blocks are written to the Report File which is then routed to Building 17, DTNSRDC to be printed with a high-speed printer (31).

His work completed for now, the user leaves CDMS (32) and "logs out" of INTERCOM (33).

APPENDIX D

SUMMARY OF CDMS STORAGE CAPACITIES

MAXIMUM DATA BLOCK STORAGE CAPACITIES:

Number of data blocks on each file...............2**53-1

Number of words in each data block...............262,143

Number of blocks that may be saved at any one time for reallocation.............................. 81,090

Number of full or partial data blocks that may be contained in buffer at any one time............ 20

MAXIMUM INVERTED LIST STORAGE CAPACITIES:

Number of inverted element lists per file........ 100

Number of entries per inverted list element......173,740

Number of qualifying inverted list entries per query..................................... 64,897

MAXIMUM BLOCK TYPE STORAGE CAPACTIES:

Number of block types on each file............... 100

Number of data elements per block type........... 250

Number of subblocks per block type............... 8

Number of repeating groups per subblock.......... 14

Number of elements per repeating group........... 50

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX E

USE OF CDMS IN BATCH ENVIRONMENT

The COMRADE Executive was originally designed and implemented as an interactive system. There arose, however, a need to be able to perform large non-interactive jobs while operating under COMRADE. Thus a batch capability was added to the Executive.

The incorporation of a batch mode capability forced several significant changes in the format of user input, data and output results, the major one being the grouping of input data as logical card records, one logical record for each executed program that requires data. One record (the first record) to indicate names of the procedures to be executed by the Executive must always be included as well as any input required by the PDL programs. In the case of CDMS, one additional data record, that required by the CDMS program, must be included.

The input to CDMS consists of a series of cards containing the CDMS commands to be performed. Commands in batch have the same syntax as those issued interactively, the only difference being that, under the interactive mode, the user may be consulted about continuing processing or file disposition. CDMS, in batch mode, assumes the user knows what he wants, and attempts to respond fully to each command.

A sample run follows:

```
job card              (must have at least CM 61000)
charge card
COMRADE.
7/8/9
CDMS
SIGNOFF
7/8/9
ATTACH,CDMSDB,PRESIDENTSDATABASE,ID=COMR.
DATABASE,CDMSDB
STATS
HELP
BTNAMES
DEFINE
PRINT PRES
EXIT
6/7/8/9
```

REFERENCES

1. Control Data Corporation, "SCOPE Reference Manual, 6000 Version 3.4," Publication No. 60307200.

2. Wallace, M. et al., "COMRADE Data Storage Facility Users Manual," David W. Taylor Naval Ship Research and Development Center Report 76-0003.

3. Gorham, W. et al., "COMRADE Data Management System Host Language Interface Users Manual," David W. Taylor Naval Ship Research and Development Center Report 76-0006.

4. Control Data Corporation, "INTERCOM Reference Manual, 6000 Version 4," Publication No. 60307100.

5. CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," CODASYL Systems Committee Report (May 1971).

6. Fong, E. et al., "Six Data Base Management Systems: Feature Analysis and User Experience," National Bureau of Standards Technical Note 887 (Nov 1975).

7. Wallace, M., "COMRADE Absolute Subroutine Utility Users Manual," David W. Taylor Naval Ship Research and Development Center Report 76-0004.

8. Control Data Corporation, "COBOL Version 4 Reference Manual," Publication No. 60384100.

9. Control Data Corporation, "FORTRAN Extended Version 4 Reference Manual," Publication No. 60305601.

10. Control Data Corporation, "COMPASS Version 3 Reference Manual," Publication No. 60360900.

11. Rhodes, T. and W. Gorham, "COMRADE - The Computer-Aided Design Environment Project - an Introduction," David W. Taylor Naval Ship Research and Development Center Report 76-0001.

12. Tinker, R. and I. Avrunin, "COMRADE Executive System Users Manual," David W. Taylor Naval Ship Research and Development Center Report 76-0002.

AD-A046 818 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
COMRADE DATA MANAGEMENT SYSTEM CONVERSATIONAL INTERFACE USERS M--ETC(U)
JAN 76 W C GORHAM, S E WILLNER, R J MARTIN

UNCLASSIFIED DTNSRDC-76-0007 NL

2 OF 2
AD
A046818

END
DATE
FILMED
12-77
DDC

INITIAL DISTRI

Copies	
1	U.S. Army Electronics Comman
1	CHONR 437/Denicoff
1	NAVSEA 6E24/Chaiken
25	NAVSEC:
	1 SEC 6105B/Dietrich
	20 SEC 6105B/Berry
	1 SEC 6114B2/Fuller
	1 SEC 6133E/Straubinger
	1 SEC 6152D/Lee
	1 SEC 6179A20/Singer
12	DDC
1	NASA-Langley/R. Fulton
1	Grumman Aerospace Corp./Will

CENTER DISTRIB

Copies	
1	18/1808 Gleissner
1	1802.2/Frenkiel
1	1802.4/Theilheimer
1	1805/Cuthill
2	1809/Harris
1	182/Camara
1	1822/Rhodes
1	1822/Wallis
1	1824/Zaritsky
20	1828/Gorham
1	1826/Culpepper

CENTER DISTRIBUTION (Continued)

Copies	
30	5211 Reports Distribution
1	5221 Library--Carderock
1	5222 Library--Annapolis
1	712.9/Pierce

DTNSRDC ISSUES THREE TYPES OF REPORTS

(1) DTNSRDC REPORTS, A FORMAL SERIES PUBLISHING INFORMATION OF PERMANENT TECHNICAL VALUE, DESIGNATED BY A SERIAL REPORT NUMBER.

(2) DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, RECORDING INFORMATION OF A PRELIMINARY OR TEMPORARY NATURE, OR OF LIMITED INTEREST OR SIGNIFICANCE, CARRYING A DEPARTMENTAL ALPHANUMERIC IDENTIFICATION.

(3) TECHNICAL MEMORANDA, AN INFORMAL SERIES, USUALLY INTERNAL WORKING PAPERS OR DIRECT REPORTS TO SPONSORS, NUMBERED AS TM SERIES REPORTS; NOT FOR GENERAL DISTRIBUTION.